Titre: Title:	Conception d'une API pour le support de la mobilité avec IPV6
Auteur: Author:	Vincent Passelande
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Passelande, V. (2004). Conception d'une API pour le support de la mobilité avec IPV6 [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7428/

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UNE API POUR LE SUPPORT DE LA MOBILITÉ AVEC IPV6

VINCENT PASSELANDE
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2004

Your file Votre référence
ISBN: 0-494-01378-8
Our file Notre référence
ISBN: 0-494-01378-8

UNIVERSITÉ DE MONTRÉAL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

CONCEPTION D'UNE API POUR LE SUPPORT DE LA MOBILITÉ AVEC IPV6

présenté par : PASSELANDE Vincent
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. DESMARAIS Michel, Ph.D., président
M. PIERRE Samuel, Ph.D, directeur de recherche et membre
M. QUINTERO Alejandro, Doct., membre

REMERCIEMENTS

Je tiens à remercier M. Samuel Pierre, mon directeur de recherche, pour son soutien tout au long de ma recherche. De même, je remercie M. Laurent Marchand et toute l'équipe du Centre de recherche LMC de Ericsson Canada pour les conseils et toute l'aide qu'ils m'ont apportée.

Je voudrais également remercier mes parents, mes frères et sœur et mes amis pour m'avoir toujours supporté.

RÉSUMÉ

L'émergence des réseaux de troisième génération et l'amélioration constante des routeurs nous a mené à nous pencher sur la séparation des routeurs en une couche contrôle (CE) et une couche transport (FE). La suite des protocoles MIPv6 est sur le point de devenir des standards à l'IETF. Le niveau contrôle s'occupe de nombreux protocole et fonctionnalités. Ce niveau logiciel permet des changements si le besoin s'en fait sentir. Le niveau transport est plus matériel. Chaque année, de nouveaux processeurs réseaux pour le traitement des paquets sortent; ils sont de plus en plus rapides et de moins en moins chers. La séparation en deux niveaux distincts permet une évolution séparée. Cette séparation permet une mise à l'échelle indépendante. L'évolution séparée entre les deux couches implique la création d'une API permettant de faire le lien entre ces couches. La conception d'une API permet également de pouvoir changer une carte (FE ou CE) sans avoir à modifier le reste du routeur. Mon travail s'attarde sur une API pour la gestion de la mobilité.

La gestion de la mobilité se base sur le protocole MIPv6. Cependant, ce protocole ne permet pas de supporter du trafic temps réel ou du trafic sensible au délai. Un usager se connecte au réseau à travers un routeur d'accès. À chaque fois qu'il se déplace (qu'il change de routeur d'accès), il est obligé de configurer une nouvelle adresse IP. Ce changement d'adresse implique une perte de la connexion au niveau de la couche trois du modèle OSI. Pour palier ce problème, on inclut à MIPv6 une suite de protocoles. L'API va donc couvrir MIPv6, HMIPv6, FMIPv6 et le « bicasting ».

HMIPv6 ajoute un niveau hiérarchique au réseau MIPv6 ce qui permet de réduire la signalisation. FMIPv6 anticipe la relève permettant de réduire la perte de paquets. Le « bicasting » rend possible à l'usager de recevoir des paquets par plusieurs routeurs d'accès, ce qui aide à éviter la perte de paquets.

Deux API vont être définis dans ce mémoire. Le premier traite de toutes les fonctionnalités inhérentes aux différents protocoles utilisés. L'API de gestion de mobilité définit un ensemble de structures et de fonctions de manière à maintenir l'ensemble des informations contenues au niveau du FE. Il définit ce qui doit être

implémenté dans les différents routeurs du réseau autant au niveau du routeur d'accès, du MAP (« Mobility Anchor Point ») et du HA (« Home Agent »). Le deuxième traite de la gestion des interfaces. L'API de gestion d'interface définit la manière selon laquelle les interfaces de la couche trois sont associées aux tables locales de routage du FE ainsi que la manière selon laquelle les adresses IP sont associées à une interface de la couche trois.

La validation de l'API a nécessité une implémentation. Nous avons utilisé le code MIPL (« Mobile IP for Linux ») développé à l'Université d'Helsinki. Nous avons modifié ce code pour le séparer en couches CE et FE et ainsi implémenté l'API. Cette intégration nous a permis de vérifier la viabilité de notre proposition.

ABSTRACT

The emergence of third generation mobile networks and the constant improvement of routers lead to the separation of the routers in a layer controls (CE) and a layer transport (FE). The MIPv6 protocol suite are about to become standards at the IETF. The control layer deals with many protocols and functionalities. This level is software implemented allowing changes if needed. The transport layer is material. Each year, new network processors for packet processing are released; they are becoming more and more faster and less expensive. Separation in two distinct layers allows a separate evolution and independent scaling. The separation in two layers implies the creation of API making it possible to establish the link between these layers. The design of an API also makes it possible to be able to change a card (FE or CE) without having to modify the router. My work focuses on an API for the management of mobility.

The mobility management is based on MIPv6 protocol. However, this protocol does not make it possible to support real time traffic or sensitive delay traffic. A user connects himself to the network through an access router. Each time that it moves (change of access router), it is obliged to configure a new IP address. This change implies a loss of connection on the layer tree of the OSI model. To resolve this problem, we add to MIPv6 a protocol suite. The API will cover MIPv6, HMIPv6, FMIPv6 and the bicasting.

HMIPv6 adds a hierarchical level to the MIPv6 network making possible to reduce signaling. FMIPv6 anticipates the hand-off making possible to reduce the packet loss. The bicasting makes possible to the user to receive packets by several access routers and helps avoid the packet loss.

Two API are defined in this memory. The first one focuses on all the inherent functionalities to the various protocols used. The management mobility API defines a set of structures and functions to maintain the information on the FE layer. It defines what must be implemented in various routers such as access router, MAP (Mobility Anchor Point) and HA (Home Agent). The second API focuses on the management of the interfaces. The interface management API defines the way layer three

interfaces are associated with the local routing tables as well as the manner according IP addresses are associated with layer three interface.

The API validation requires an implementation. We used MIPL code (Mobile IP for Linux) developed at Helsinki University. We have to modify this code to separate it in CE and FE layers and thus implement the API. This integration enables us to verify the viability of our proposal.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTES DES SIGLES ET ACRONYMES

AR	routeur d'accès (« Access Router »)
Binding	Association de l'adresse nominale du MN avec son CoA
BU	message de binding (« Binding Update »)
CN	nœud correspondant (« Corresponding Node »)
CoA	adresse unicast associée au MN (« Care-of-address »)
DAD	détection de duplication d'adresse (« duplication address detection »)
FMIPv6	MIPv6 avec relève rapide (« Fast handover MIPv6 »)
HA	Agent sur le réseau d'origine (« Home Agent »)
HMIPv6	MIPv6 avec hiérarchie (« Hierarchical MIPv6 »)
LBU	BU local (« Local BU »)
LCoA	CoA local, indique la position courante du MN (« On-Link CoA »)
MAP	HA local (« Mobile Anchor point »)
MIPv6	IP mobile version 6 (« Mobile IPv6 »)
MN	nœud mobile (« Mobile Node »)
RA	message avertissement de routeur (« Routeur Advertisement »)
RCoA	CoA régional, sous le réseau du MAP (« Regional CoA »)
DAD-free NCoA	un NCoA dont l'unicité est déjà garantie
OS	système d'exploitation

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

Depuis l'émergence des réseaux cellulaires, on dénombre de plus en plus d'usagers et leur nombre augmente toujours de façon significative. On peut se demander dans quelle mesure les réseaux IP (« Internet Protocol ») seront capables de répondre à la demande en terme de nombre de clients et de qualité de service que ces derniers vont exiger, tout en évitant la congestion du réseau. Pour accommoder un nombre grandissant d'usagers et de périphériques, on doit recourir à la version 6 de IP (IPv6). La version 4 de IP est d'ores et déjà exclue car le nombre d'adresses disponibles est insuffisant. Mais, IPv6 mobile (MIPv6) ne supporte pas le trafic en temps réel. De ce fait, il ne peut pas être utilisé seul et doit être associé à d'autres protocoles complémentaires. De plus, les routeurs doivent être capables de traiter un grand nombre d'appels et de messages de signalisation. Il faut donc les repenser pour qu'ils perdent leurs aspects monolithiques et deviennent plus évolutifs tout en améliorant leur efficacité. L'objectif de ce mémoire est de pouvoir incorporer une version MIPv6 améliorée à un nouveau genre de routeur. Cette version du protocole doit être capable de supporter des relèves rapides et sans coupures quelle que soit la situation. Dans ce chapitre d'introduction, nous allons tout d'abord présenter les concepts de base sur lesquels reposent les éléments de notre problématique. Ensuite, nous verrons les éléments de problématique reliés au sujet et nous formulerons nos objectifs de recherche. Finalement, nous présenterons un plan du mémoire.

1.1 Définitions et concepts de base

Les architectures traditionnelles des routeurs ne permettent pas la flexibilité ni l'évolutivité nécessaire au niveau des protocoles utilisés. De plus, elles ne

possèdent généralement pas la rapidité nécessaire pour les traitements à effectuer en temps réel. Les architectures les plus populaires que l'on retrouve sont basées sur des circuits intégrés de type ASIC. Cette technologie permet d'obtenir des éléments très rapides qui ne sont par contre ni flexibles ni évolutifs. Leurs possibilités de programmation demeurent relativement restreintes. À l'inverse, il existe des architectures basées sur des processeurs standards. On peut y intégrer facilement toutes les parties logicielles que l'on désire. Cela rend ces systèmes très flexibles. Par contre, ils ne permettent pas d'obtenir des niveaux de performance adéquats, surtout en considérant un nombre d'usagers (et donc de signaux) grandissant.

L'architecture basée sur les processeurs réseau est un compromis entre les deux technologies précédentes. Ces processeurs n'implémentent des piles de protocoles utilisées dans les nœuds que les parties qui requièrent un accès direct aux données. Toutes les parties de contrôle peuvent se faire séparément avec un logiciel adapté. Cette technologie est la plus appropriée pour obtenir flexibilité, évolutivité et rapidité.

La version 4 du protocole IP est inadéquate en ce qui concerne le nombre d'adresses disponibles, la sécurité, etc. Nous avons donc choisi d'utiliser IPv6 et plus précisément MIPv6, pour la gestion de la mobilité. Il se pose tout de même un problème car MIPv6 ne supporte pas le temps réel. Or, si nous voulons parler de vidéoconférence, il faut supporter le temps réel, des relèves sans coupure ni perte de paquets ou le moins possible. Par exemple, lorsque l'on change de routeur d'accès, on est obligé de changer d'adresse IP. Le changement d'adresse IP entraîne une perte de la connexion au niveau de la couche 3 du modèle OSI (« Open System Interconnection ») et donc une perte de connexion. Le temps de retrouver cette connexion, la communication a été coupée et des paquets ont été perdus. Il faut trouver des mécanismes permettant que ce genre d'évènement se produise sans perte de paquets. Nous devons donc ajouter des améliorations à MIPv6 pour pallier ce problème.

1.2 Éléments de la problématique

Comme on l'a vu, les routeurs actuels ont des architectures logicielles ou matérielles qui ont des problèmes intrinsèques. Les premières sont évolutives mais lentes, alors que les secondes sont rapides mais chères et non évolutives. L'évolution des routeurs nous pousse à nous intéresser à la structure logique de cet élément. Cette structure se compose de plusieurs entités qui coopèrent pour fournir une fonctionnalité donnée. Deux types de composantes principales existent : le *niveau contrôle* et le *niveau acheminement*. La Figure 1.1 en est une illustration.

En général, le *niveau acheminement* effectue toutes les opérations de transmission de données. Le niveau contrôle fournit les fonctionnalités de commande, de traitement et de gestion des protocoles d'acheminement et de signalisation, agit au dessus et est responsable des opérations de contrôle, il s'occupe des protocoles de routage et de signalisation et dicte le comportement du niveau *acheminement* en manipulant les tables de routage, la QoS (Qualité de service) et des listes de contrôle d'accès. Basé sur l'information acquise, le niveau contrôle dicte le comportement pour l'expédition des paquets du niveau *acheminement* par l'intermédiaire du protocole d'interconnexion.

Un ensemble de mécanismes pour relier ces composantes fournit une évolutivité accrue et permet aux deux éléments d'évoluer indépendamment. Ces composantes, bien qu'agissant ensemble, exécutent des fonctions en grande partie indépendantes l'une de l'autre.

Le niveau *contrôle* possède l'intelligence nécessaire à la gestion et à la configuration du routeur et est généralement de type logiciel. Le niveau *acheminement* doit être le plus simple possible pour que toute sa capacité soit utilisée pour le traitement et l'expédition des paquets. Ce niveau est généralement de type matériel.

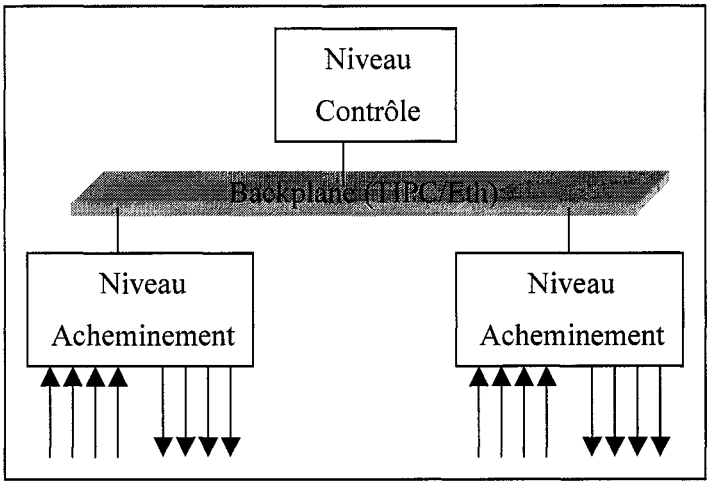


Figure 1.1 Blocs logiques d'un système

Un système peut être représenté dans une seule et même boîte mais il peut également être décomposé en deux parties distinctes. Dans ce cas, chaque partie appartient à des ensembles physiques distincts. La Figure 2 montre un cas où l'on veut que les différentes fonctions concernant la mobilité soient réunies dans un serveur pouvant contrôler différents systèmes comprenant le niveau acheminement. Cela permet de pouvoir changer les fonctionnalités de commutation du niveau acheminement sans avoir à faire de modification dans le niveau contrôle. Puisque les protocoles, une fois acceptés, ne changent pas, le niveau contrôle ne devrait pas changer significativement. Mais si cela est nécessaire, les améliorations sont toujours possibles.

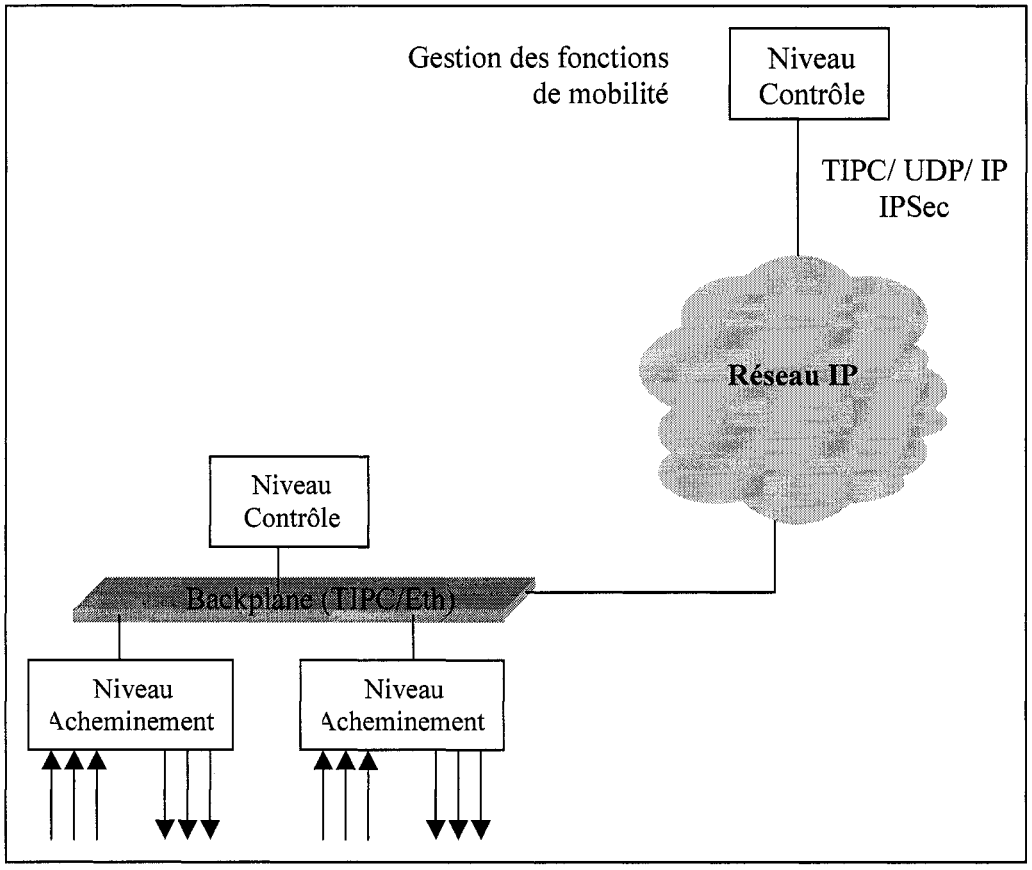


Figure 1.2 Blocs logiques d'un système réparti

1.3 Objectifs de recherche

Ce mémoire a pour objectif de développer une interface de programmation (API) pour les niveaux *contrôle* et *acheminement* et ce, pour les différents nœuds d'un réseau IP. Pour cela, il faut :

- analyser le fonctionnement du protocole MIPv6 et voir comment y intégrer des protocoles qui permettent de l'améliorer;
- séparer les différentes fonctionnalités (pour toutes les opérations et pour tous les nœuds) à effectuer entre le *niveau acheminement* et le *niveau contrôle*;
- concevoir et implanter un API pour la gestion de la mobilité;
- quantifier l'amélioration apportée à un réseau IP.

1.4 Plan du mémoire

Ce mémoire traite de la conception d'un API pour la gestion de la mobilité dans les réseaux IP et se divise en cinq chapitres distincts.

Le second chapitre s'attarde à la revue de littérature. Pour pouvoir traiter le sujet correctement, nous devons connaître ce qui a déjà été fait. Ce chapitre se divise en deux parties. Nous allons tout d'abord parler de la gestion de la mobilité. Pour ce faire, on doit avoir une idée de l'architecture globale du réseau pour savoir où sont placés les différents nœuds. Ensuite, nous verrons comment MIPv6 gère la mobilité et nous allons voir d'autres protocoles comme HMIPv6 ainsi que des extensions comme FMIPv6 et le « bicasting ». Nous allons ensuite parler de l'architecture générale du système, sa topologie c'est-à-dire la division entre les niveaux *contrôle* et *acheminement* ainsi que la couche transport qui fait le lien entre les deux.

Une fois que les concepts de bases ont été définis, on va pouvoir s'attaquer au troisième chapitre. En fonction que ce qui aura été dit, on pourra dégager une proposition de modèle d'API pour la gestion de la mobilité.

Dans le quatrième chapitre, nous verrons la mise en œuvre de cette proposition en faisant l'implémentation de cet API. Nous pourrons aussi, à partir de cette implémentation, faire une analyse des résultats et ainsi voir ce que ce modèle apporte de plus à un routeur.

Enfin, dans le cinquième chapitre, nous ferons une conclusion de ce mémoire. Ce dernier chapitre mettra l'accent sur les principaux résultats obtenus, sur les limitations de la conception et les améliorations possibles à apporter ainsi que sur les différentes possibilités de recherche à venir.

CHAPITRE 2

GESTION DE LA MOBILITÉ ET REQUIS DU SYSTÈME

La conception d'une API pour le support de la mobilité dans les réseaux de troisième génération nécessite une étude préalable du protocole MIPv6. Or, avec MIPv6, on est obligé de configurer une nouvelle adresse IP à chaque déplacement. Le changement d'adresse implique une perte de la connexion au niveau trois de la couche OSI. Pour diminuer la perte de paquets et supporter des relèves plus rapides, d'autres protocoles peuvent être ajoutés à MIPv6. L'API doit être inséré dans des routeurs privilégiant une séparation en une couche transport et une couche contrôle. En effet, la motivation menant à la conception de l'API vient du fait que nous voulons permettre une évolution séparée de ces deux couches. La partie contrôle s'occupe de nombreux protocoles et fonctionnalités. Cette couche est plus de nature logicielle, ce qui permet de petits changements si le besoin s'en fait sentir. La couche transport est plus de nature matérielle. Chaque année, de nouveaux processeurs réseaux apparaissent sur le marché. Ils sont de plus en rapide et de moins en mois chers. La création d'un API va donc permettre de changer une carte transport ou contrôle sans avoir à apporter de changements significatifs au routeur affecté. Ce chapitre analyser, dans un premier temps, le fonctionnement des différents protocoles nécessaires à la gestion de la mobilité et, dans un deuxième temps, l'architecture générale du système où sera implémenté l'API.

2.1 Gestion de la mobilité

Comme on l'a déjà dit, la gestion de la mobilité se fait sous la version 6 du protocole IP [1]. Les raisons principales sont :

- épuisement des adresses disponibles avec IPv4;
- simplification du format de l'entête (certains champs de l'entête Ipv4 ont été enlevés ou rendus optionnels pour réduire le coût de la gestion des paquets);
- support amélioré des options et des extensions futures;
- fonctionnalité d'étiquetage de flux d'information;
- fonctionnalité d'authentification et de confidentialité;
- problèmes de sécurité (dans Ipv6, la sécurité est partie intégrante, alors quelle est ajouté dans IPv4).

Par contre, MIPv6 utilisé seul ne permet pas la communication en temps réel. Les temps de relève sont trop longs et la quantité de signalisation échangée est trop importante, entraînant des temps de latence ne permettant pas de supporter un trafic temps réel.

Selon [1], un nœud mobile (MN) est toujours joignable par deux adresses. On retrouve premièrement son adresse nominale basée sur le préfixe de sous-réseau de son réseau d'origine et deuxièmement son adresse CoA (« Care-of-address ») basé sur le préfixe du sous-réseau visité. Un MN est toujours accessible à son adresse nominale quelle que soit sa position. Quand le MN est sur son réseau d'origine, les paquets adressés à son adresse nominale lui sont acheminés directement par les mécanismes de routage Internet classiques. Quand il visite un autre réseau, il est toujours rejoignable par son adresse nominale mais peut aussi l'être par une ou plusieurs CoAs. Dans ce cas, le MN peut être rejoint par deux adresses.

Le MN transmet, aux différents nœuds avec lesquels il communique, des messages permettant de faire l'association entre ces deux adresses (association CoA⇔ adresse nominale). Quand le MN est à l'extérieur de son réseau d'origine, il enregistre son adresse CoA à son HA (« Home Agent »). Le MN peut aussi donner les informations concernant sa position aux CN (« Corresponding Node ») avec lesquels il communiquent. Cela se fait par la procédure « Correspondant Binding ».

Pour maintenir une connexion avec MIPv6, le MN transmet des BU (« Binding Update » : message d'association d'adresses) vers le HA et les CN à chaque mouvement. L'envoi de ces BU prend du temps et cause une interruption à chaque déplacement vers un nouveau routeur d'accès. Pour cette raison, un nouveau noeud a été créé, le MAP (HMIPv6 : « Hierarchical Mobile Ipv6 ») [2]. Il peut être placé à n'importe quel niveau de la hiérarchie du réseau. Le MAP permet de réduire la signalisation à l'extérieur du domaine local. Il permet au MN de transmettre les BU au MAP local plutôt qu'aux HA et CN. Seulement un BU a besoin d'être transmis par le MN avant que le trafic du HA et des CN soit réacheminé à la nouvelle position, et ceci indépendamment du nombre de CN avec lequel le MN communique. Le MAP agit comme un HA local. HMIPv6 supporte aussi les relèves rapides (FMIPv6 : « Fast Handover Mobile Ipv6 ») [3] et [4].

Lors d'un déplacement, le MN est incapable d'envoyer et de recevoir des paquets, dû au délai du changement de lien et aux opérations du protocole IP. Ce temps est souvent trop grand pour le support de trafic temps réel ou du trafic sensible au délai. FMIPv6 [3] explique comment permettre à un MN d'envoyer des paquets dès qu'il détecte un nouveau lien, et comment livrer des paquets à un MN dès que sa présence est détectée par le nouveau routeur d'accès du MN. Dans ce cas, le MN peut continuer à utiliser son ancien CoA même après avoir changé de routeur d'accès et retarde ainsi l'envoie de BU. Un tunnel est créé entre l'ancien et le nouveau routeur d'accès du MN de telle sorte que celui-ci puisse prendre un certain temps pour configurer sa nouvelle adresse CoA. Le temps supplémentaire lui permet de pouvoir préparer la relève et de perdre moins de paquets.

Cependant, quand le MN se déplace rapidement ou qu'il a un mouvement comme le ping-pong, FMIPv6 n'est pas suffisant pour éviter la perte de paquets. Une extension à FMIPv6, le « n-casting » a donc été introduite [5]. Elle permet d'éviter l'ambiguïté sur le moment précis de la relève. Avec le « bicasting », on envoie les paquets au MN via les différents AR sur lesquels il est peut-être attaché. Cette méthode permet donc de minimiser la perte de paquets

2.1.1 Architecture globale du réseau

La Figure 2.1 présente un exemple de l'architecture d'un réseau IP et permet de voir l'emplacement des différents nœuds. On peut remarquer qu'un MN est capable de se connecter au réseau via différentes interfaces d'accès.

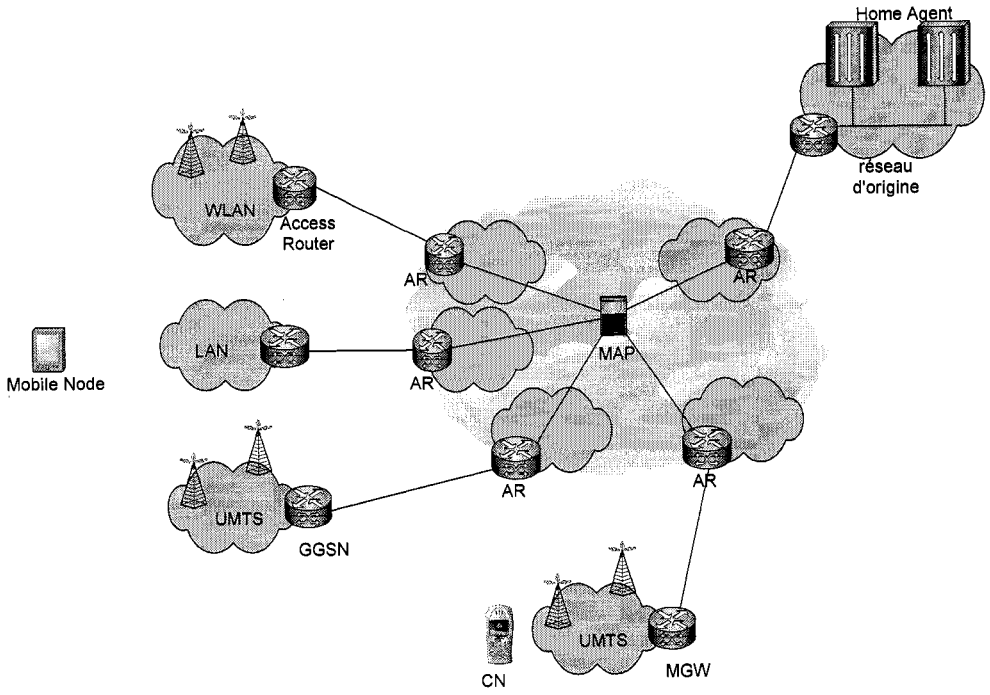


Figure 2.1 Architecture globale du réseau

Si le MN se déplace, il doit être capable de conserver sa connexion. Un changement de position implique un changement d'adresse IP du MN et une perte possible de la connexion au niveau de la couche 3 donc, une perte de communication. Regardons premièrement comment MIPv6 gère cette mobilité.

2.1.2 Gestion de la mobilité avec MIPv6 [1]

Il y a deux modes de communication entre le MN et le CN : le tunnel bidirectionnel et la route optimisée.

Le premier mode ne nécessite pas le support MIPv6 de la part du CN et est possible même si le MN n'a pas enregistré son association au CN. Les paquets du MN au CN sont envoyés à travers un tunnel jusqu'au HA (« Reverse tunnelling ») et du HA envoyé au CN. Les paquets du CN sont acheminés au HA qui les envoie à travers le tunnel à l'adresse CoA primaire du MN.

Le second mode, la route optimisée, nécessite que le MN enregistre son association d'adresse au CN. Le CN connaît ainsi l'adresse CoA du MN et est capable d'envoyer des paquets directement au MN. À l'envoi d'un paquet, le CN vérifie ses entrées d'association pour savoir s'il y a une entrée pour l'adresse de destination du paquet. Si oui, il utilise l'optimisation de route; les paquets n'ont pas besoin de transiter par le réseau d'origine. Étant donné que le chemin est plus court, la communication entres les deux nœuds est plus rapide. La Figure 2.2 illustre les deux différents modes.

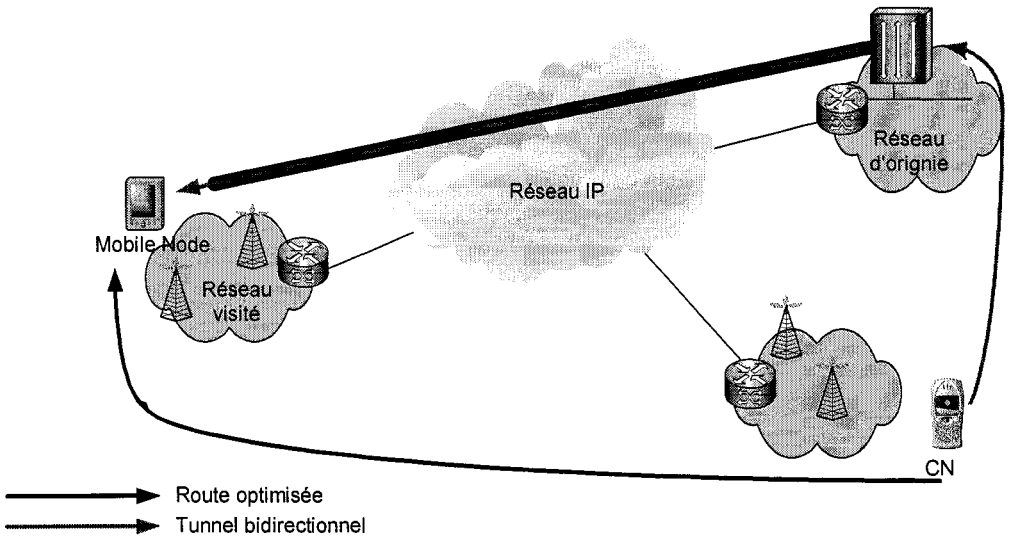


Figure 2.2 Communication entre les différents éléments

2.1.2.1 Enregistrement du CoA au du HA

Après avoir détecté un mouvement, le MN doit générer un nouveau CoA. Le MN détecte un mouvement grâce aux messages envoyés par les routeurs d'accès, les « Router Advertisements » (RA). Ces messages contiennent le préfixe du sous-réseau

visité. Le MN compare le préfixe qu'il utilise ainsi que ce préfixe reçu et peut en déduire s'il s'est déplacé. À partir de ce préfixe, il construit une CoA (qui représente sa position actuelle). Quand le MN est loin de son réseau d'origine, il enregistre son CoA primaire avec son HA. Le MN est obligé de faire cet enregistrement s'il veut être rejoint à sa position actuelle. Pour que le HA transmette les paquets jusqu'au MN, il a besoin de connaître son adresse courante. Le seul cas où cet enregistrement est à proscrire, c'est quand le MN est sur le même réseau que son HA ; le MN n'a pas besoin que le HA intercepte les paquets pour lui car le MN s'en occupe tout seul. La Figure 2.3 illustre la séquence d'évènements lors de l'enregistrement d'une adresse au HA.

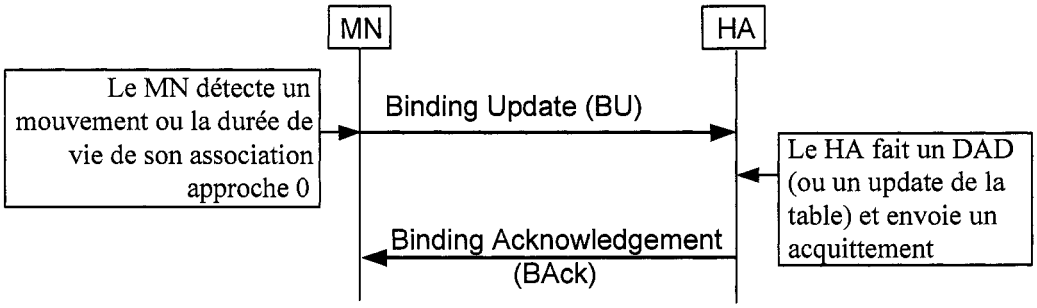


Figure 2.3 Enregistrement du CoA au HA

Le BU qu'envoie le MN contient le CoA proposé par le MN. Ce CoA est proposé mais il n'est pas dit que ce soit celui qui va être utilisé. Sur réception de ce BU, le HA doit faire une détection de duplication d'adresse (DAD : « duplication adress detection »). Cela sert à vérifier qu'aucun autre HA sur le réseau d'origine n'intercepte pas déjà des paquets pour cette adresse. Si la détection de la duplication d'adresse échoue, les nœuds doivent cesser d'employer l'adresse et attendre la reconfiguration d'une nouvelle adresse. L'association ne pourra pas se faire tant que le CoA sera déclaré invalide. L'acquittement (BAck : « Binding Acknowledgement ») indique si l'association a réussi et, dans le cas où elle a échouée, pourquoi. Une durée de vie est reliée à l'association d'adresse. Le MN est donc obligé d'envoyer régulièrement des messages d'association d'adresse.

Si le HA accepte le BU, il doit créer une nouvelle entrée en « Binding cache » pour l'association d'adresse ou mettre à jour celle concernée. Il doit marquer cette entrée comme étant un enregistrement primaire. Les entrées marquées comme enregistrement primaire doivent être exclus de la politique normale de remplacement de « Binding cache » et ne doivent pas être enlevées tant que la durée de vie n'a pas expiré. À partir de ce moment, le HA doit intercepter les paquets sur le réseau d'origine pour les transmettre au MN et doit être prêt à accepter les paquets en tunnel inverse venant du CoA du MN.

Tant qu'un nœud sert de HA, il utilise la découverte de voisin [9] (« IPv6 Neighbor Discovery ») pour intercepter les paquets unicast adressés au MN. Pour ce faire, le HA doit agir comme proxy pour le MN et répondre à tous les « Neighbor Sollicitation » pour lui. Quand le HA reçoit un « Neighbor Sollicitation », il doit regarder si l'adresse cible spécifiée dans ce message correspond à l'adresse du MN (qui s'est enregistré auprès de lui). Si c'est le cas, le HA doit répondre à la sollicitation avec un « Neighbor Advertisement » donnant l'adresse MAC du HA comme si c'était celle correspondant au nœud ayant l'adresse cible (MN). Agissant comme proxy, cela permet aux autres nœuds sur le lien du HA de résoudre l'adresse du MN et permet au HA de défendre ses adresses (celles en « Binding cache ») pour le DAD et de faire une détection de duplication d'adresse. Quand un nœud agit comme HA pour un MN, il doit intercepter les paquets pour le MN et les lui transmettre via un tunnel en utilisant l'encapsulation IPv6 (« inner IP »). Quand le MN reçoit ces paquets, il les décapsule et les traite normalement. Pour ce faire, quand un nœud commence à servir de HA, il doit envoyer un message « Neighbor Advertisement » sur son lien de la part du MN (adresse nominale du MN) pour donner son adresse MAC (celle du HA). Tous les champs dans le « Neighbor Advertisement » doivent être réglés de la même manière que si le MN aurait envoyé lui-même ce message. Les nœuds voisins ont ainsi l'impression que le MN est à coté d'eux mêmes si en fait il n'y a que le HA.

Une association peut devoir de se faire dé-enregistrée, par exemple, lorsque le MN retourne sur le réseau d'origine. Le MN demande au nœud récepteur de ne plus agir comme HA et d'enlever son CoA primaire en envoyant un BU spécifique. Le HA arrête alors d'intercepter les paquets pour le MN.

2.1.2.2 Enregistrement du CoA au CN

Le MN peut donner des informations sur son emplacement au CN à travers la procédure « correspondant binding ». La première partie de cette procédure est la procédure de retour de routabilité [1] (« Return Routability Procedure »). Elle est effectuée pour autoriser l'établissement de l'association d'adresse. Pour que le MN puisse entamer cette procédure, il faut qu'il soit déjà enregistré auprès d'un HA. La deuxième partie est l'envoi du BU qui est utilisé pour autoriser et transmettre l'association d'adresse.

Le MN décide d'engager ce processus avec le CN sur réception d'un paquet venant du CN mais pour lequel il n'a pas d'association ou pour la mise à jour d'une association existante.

La protection des BU envoyées aux CN n'exige pas la configuration d'associations de sécurité ou l'existence d'une infrastructure d'authentification entre les MN et les CN. À la place, on utilise la procédure de retour de routabilité pour s'assurer que le bon MN envoie le message. L'intégrité et l'authenticité des BU envoyés au CN sont protégées en utilisant un algorithme de verrouillage des informations parasites utilisant la clef de gestion d'association, Kbm (« binding management key »). Cette clef est établie grâce à un échange de données lors de la procédure de retour de routabilité.

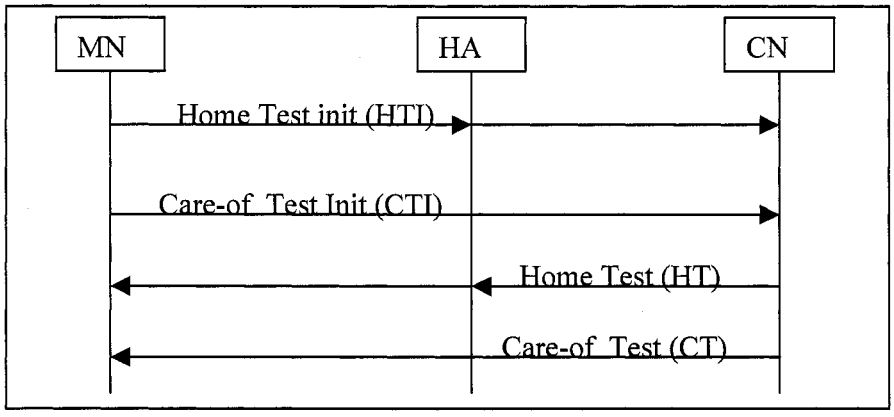


Figure 2.4 Procédure de retour de routabilité

La procédure de retour de routabilité, illustrée à la Figure 2.4, permet au CN d'obtenir une assurance raisonnable que le MN est joignable par son CoA et son adresse nominale. C'est seulement avec cette assurance que le CN accepte les BU du MN. Cette assurance est prise en testant si les paquets envoyés aux deux prétendues adresses arrivent au MN. Le MN passe les tests s'il prouve qu'il a bien reçu certaines données. Lors de cette procédure, le « Care-of-init keygen token » et le « Home-init token », sont envoyés au CN et reviendront plus tard au MN. Ces données sont combinées par le MN en une clef, le kbm, qu'on utilisera pour transmettre le BU. Pour chaque message « Home » ou « Care-of-Test-Init », le MN doit générer une nouvelle valeur pour les « Care-of-init keygen token » et « Home-init token ». Ces données servent à vérifier si les messages « Home-Test » et « Care-of-Test » concordent avec les messages « Home-Test-Init » et « Care-of-Test Init ».

Home-Test-Init : Le MN envoie ce message au CN pour acquérir le « home keygen token ». Il possède comme paramètre le « home init cookie ». Ce message donne l'adresse nominale du MN au CN. Le MN doit se rappeler des valeurs des « cookies » pour obtenir une certaine assurance que ses messages seront traités par le CN désiré. Si le MN a récemment utilisé les « home token », « care-of keygen token » et « nonces indices » associées à l'adresse désirée, il peut les réutiliser. La procédure dans certains cas peut être complétée par seulement une paire de messages. Elle peut même être complétée sans message du tout si le MN a récemment utilisé le « home keygen token » et avait précédemment utilisé le même CoA, auquel cas il a aussi un « care-of keygen token » récent. Si le MN envoie un BU pour effacer une entrée, l'envoi du « Home Test Init » est suffisant.

Care-of-Test Init : Le MN envoie ce message au CN pour acquérir le « care-of keygen token ». Il possède comme paramètre le « care-of init cookie ». Ce message donne l'adresse CoA du MN au CN ainsi que le « care-of init cookie » que le CN devra retourner.

Home Test : Il est envoyé en réponse au « Home Test Init » et possède comme paramètre le « home init cookie », le « home keygen token » et le « home nonce index ». Quand le CN reçoit le « Home Test Init », il génère le « home keygen

token » qui permettra de savoir si le MN peut recevoir des messages envoyés à son adresse nominale.

Care-of Test : Il est envoyé en réponse au « Care-of Test Init » et possède le « care-of init cookie », le « care-of keygen token » et le « care-of nonce index » en paramètre. Quand le CN reçoit le « Care-of Test Init », il génère un « care-of keygen token ». Le MN enregistre le « Care-of Nonce Index » et le « care-of keygen token » dans sa liste de BU. Si l'entrée dans cette liste ne contient pas de « home keygen token», le MN doit attendre pour des messages additionnels.

Quand le MN a reçu le « Home Test » et le « Care-of Test », la procédure de retour de routabilité est complétée. Le MN possède les données dont il a besoin pour envoyer un BU au CN. Cette procédure est illustrée à la Figure 2.5. Le MN forme le Kbm avec le home keygen token et le care-of keygen token.

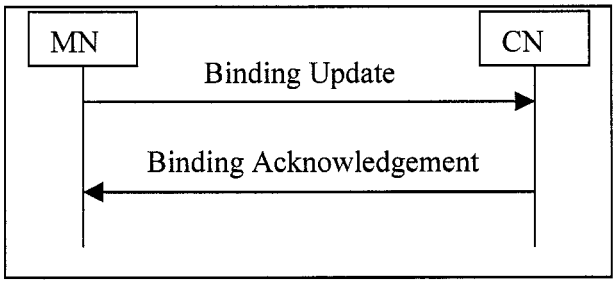


Figure 2.5 Procédure d'enregistrement MN<->CN

Binding Update : Il est utilisé pour autoriser et transmettre une association d'adresse. Ce message peut avoir deux utilités :

1. Si la durée de vie n'est pas zéro et que le CoA n'est pas égal à l'adresse nominale, alors il s'agit d'une demande pour mettre en mémoire cache l'association d'adresse du MN.
2. Si la durée de vie est de zéro ou le CoA spécifié correspond à l'adresse nominale, alors il s'agit d'une demande pour effacer l'association du MN.

Binding Acknowledgement (BAck) : Le BU est dans certains cas acquitté par le CN suivant les informations contenues dans le BU.

2.1.2.3 Mise à jour des associations

Si un émetteur sait qu'une entrée en « binding cache » est sur le point de se faire effacer mais que les communications ne sont pas terminées et qu'on veut continuer à utiliser la route optimisée, il peut envoyer un BRR (« Binding Refresh Request ») au MN pour mettre à jour une entrée. Quand un MN reçoit un message avec cette requête, il met à jour son association d'adresse. Si le MN veut enlever cette entrée (il s'est déplacé), il peut soit ignorer ce message et attendre que la durée de vie expire, soit effacer l'entrée en envoyant un BU avec une durée de vie à zéro et un CoA à la valeur de son adresse nominale. Si le MN ne sait pas s'il veut conserver l'entrée, il prend une décision basée sur différents critères comme la disponibilité des ressources.

Comme on peut le constater, énormément de messages sont échangés. À chaque déplacement et à chaque fois que la durée de vie d'une association d'adresse expire, on doit retransmettre des messages de mise à jour. Pour réduire la signalisation, une extension a été ajoutée, HMIPv6.

2.1.3 HMIPv6

HMIPv6 [2] utilise un nouveau nœud, le MAP (« Mobility Anchor Point »). Ce nœud agit comme un HA local et permet de réduire la signalisation. Avec le MAP, on ajoute un niveau hiérarchique. Le MN transmet les BU au MAP local plutôt qu'aux HA et CN. Un seul BU a besoin d'être transmis par le MN avant que le trafic du HA et des CN soit acheminé à la nouvelle position. Ceci est indépendant du nombre de CN communiquant avec le MN. Le MAP agit comme un HA local, il reçoit les paquets destinés au MN, les encapsule et les transmet au MN.

Ce protocole nécessite de nouvelles adresses, le RCoA et le LCoA. Le RCoA (« Régional CoA) est une adresse obtenue par le MN sur le réseau visité. Elle est configurée par le MN sur réception des options MAP contenues dans les RA. Cette adresse représente le domaine MAP. Le LCoA (On-Linl CoA) est configurée à partir du préfixe du routeur d'accès. Cette adresse représente la position courante du MN. Un MN entrant dans un domaine MAP reçoit un message « Router Advertisement» contenant l'information sur un ou plusieurs MAP (« MAP option »). Le MN associe

sa position locale (LCoA) avec une adresse sur le réseau du MAP (RCoA) auprès du MAP. Cette association est similaire à celle effectuée au niveau du HA sauf qu'au lieu d'associer le CoA avec l'adresse nominale, on associe le LCoA au RCoA. L'association d'adresse au niveau du HA et du CN associe maintenant le RCoA avec l'adresse nominale.

Si le MN change de position à l'intérieur du domaine MAP, seul le LCoA change. Le MN doit seulement enregistrer sa nouvelle adresse au MAP. Le RCoA ne change pas tant que le MN reste dans le domaine MAP, donc aucun BU n'a besoin d'être envoyé ni au HA ni au CN. Comme les CN et HA ne connaissent que le RCoA, il n'y a pas de changements à y faire. Cela rend le MN transparent pour le CN avec lequel il communique.

Pour utiliser la largeur de bande de manière efficace, le MN peut s'enregistrer à plusieurs MAP simultanément et peut utiliser chaque MAP (dans le même domaine) pour différents groupes de CN par exemple.

2.1.3.1 Enregistrement au MAP et au HA

L'enregistrement auprès du MAP et du HA est illustré à la Figure 2.6.

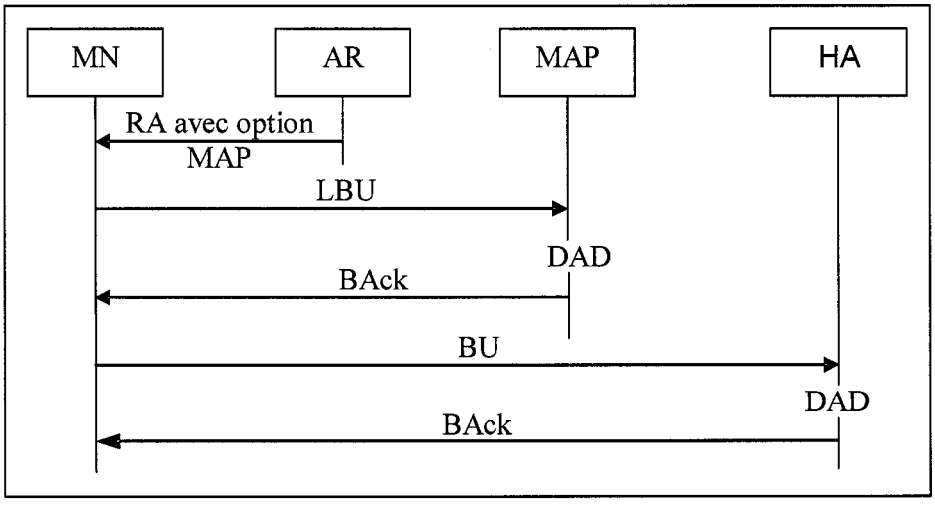


Figure 2.6 Enregistrement au MAP et au HA

Quand le MN entre dans un domaine MAP, il doit configurer le RCoA et le LCoA. Cette procédure d'enregistrement se passe en cinq étapes :

1. Le MN arrive dans un domaine MAP, il reçoit un RA (option MAP), détecte un nouveau domaine MAP et forme son RCoA et son LCoA ;
2. Le MN transmet un BU local au MAP associant le LCoA au RCoA ;
3. Le MAP fait un DAD pour voir si le LCoA est déjà utilisé ou non ;
4. Le MAP retourne un BAck indiquant le succès ou l'échec de l'association ;
5. Si l'enregistrement au MAP s'est bien effectué, le MN enregistre son RCoA au HA spécifiant l'association RCoA avec l'adresse nominale du MN (comme dans MIPv6). Le MN peut aussi transmettre un BU similaire au CN.
6. Le HA fait un DAD pour voir si le LCoA est déjà utilisé ;
7. Le #HA retourne un BAck indiquant le succès ou l'échec de l'association.

Pour améliorer la vitesse de la relève entre MAP, le MN peut transmettre un BU local à son ancien MAP en lui spécifiant son nouveau LCoA. Ce vieux MAP pourra transmettre les paquets lui arrivant au nouveau LCoA. Ceci est possible uniquement si l'ancien et le nouveau MAP sont accessibles en même temps. Si le MAP tombe en panne et plus rien ne marche, il peut donc être préférable de s'enregistrer à plusieurs MAP si possible.

L'enregistrement de l'association d'adresse au CN est tout à fait identique à celui décrit avec MIPv6.

Malgré les améliorations proposées, il n'en demeure pas moins que, quand le MN se déplace, il change d'adresse IP et perd sa connexion. HMIPv6 diminue la signalisation à l'extérieur du domaine MAP mais n'accélère pas la relève. Une nouvelle extension est donc proposée pour minimiser le délai, il s'agit de FMIPv6 [3] (« Fast handover for Mobile IPv6 »).

2.1.4 Relève rapide (FMIPv6)

FMIPv6 [3] minimise le délai de relève au niveau de la couche 3. Ce mécanisme permet l'anticipation de la relève pour que le trafic soit redirigé vers la

nouvelle position du MN avant même qu'il n'y arrive. Ce protocole permet la mise en place d'un tunnel entre deux routeurs d'accès (le précédent et le nouveau) permettant au MN d'envoyer et de recevoir des paquets même s'il n'a pas configuré une nouvelle adresse CoA et n'a pas transmis de BU. L'établissement de ce tunnel peut être initié par le MN demandant une relève ou par le réseau (PAR : « Previous Access Router »). Il y a trois phases dans les opérations :

1. Initiation de la relève ;
2. Établissement du tunnel ;
3. Transmission des paquets.

2.1.4.1 Initiation de la relève

Dans ce mécanisme, on anticipe la relève au niveau de la couche trois par des messages provenant de la couche deux. Tout commence sur un événement tel la décision par le MN d'une relève à un nouveau point d'attache. Typiquement, le MN répond à cet événement en demandant à son ancien AR (PAR : « Previous Access Router ») de l'assister dans la relève. Il fait cette demande en envoyant un « Proxy Router Solicitation » (RtSolPr) dans lequel il inclut un identificateur de la couche 2 de son point d'attache éventuel. En réponse, le PAR envoie un « Proxy Router Advertisement » (PrRtAdv) contenant l'adresse MAC, les informations de préfixe de réseau du NAR et le NCoA.

Quand le PAR initie la relève, il envoie un message PrRtAdv, sans que le MN ait envoyé de message RtSolPr, avec les paramètres nécessaires au MN pour envoyer des paquets et pour se construire une nouvelle CoA. Ensuite, le PAR prépare l'établissement du tunnel.

2.1.4.2 Établissement du tunnel

Quand le PAR reçoit un RtSolPr, il envoie un « Handover Initiate » (HI) au nouveau routeur d'accès (NAR « *New Access Routeur* »). Ce message a deux objectifs :

1. Initier l'établissement d'un tunnel bidirectionnel entre les deux routeurs d'accès pour que le MN puisse continuer à utiliser son PCoA (« Previous CoA ») ;
2. Vérifier si le NCoA (« New CoA ») est utilisable avec le NAR.

En réponse au HI, le NAR doit :

1. Créer une route pour le PCoA permettant de transmettre les paquets au MN ;
2. Créer le tunnel entre les deux routeurs ;
3. Vérifier si le NCoA indiqué dans le HI est utilisable et s'il l'est, il doit commencer à le défendre ;
4. Envoyer le HACK.

Quand le PAR reçoit le HACK, un tunnel bidirectionnel est établi entre les deux routeurs d'accès. Puisque le MN ne peut pas employer le NCoA tant qu'il ne l'a pas mis à jour avec son HA, son MAP et ses correspondants, on lui permet d'utiliser le PCoA. Le NAR « tunnelle » vers le PAR, les paquets qui ont comme adresse source le PCoA et le PAR « tunnelle » au NAR, les paquets ayant comme adresse de destination le PCoA.

2.1.4.3 Transmission de paquets dans le tunnel

Sur réception du PrRtAdv, le MN envoie un message de mise à jour d'association (FBU « *Fast Binding Update* ») au PAR. Il peut également l'envoyer après s'être attaché au NAR. Ce message fait l'association entre le PCoA du MN et l'adresse IP du NAR pour que les paquets arrivant au PAR soient tunnelés vers le NAR. Le PAR répond avec un « Fast Binding Acknowledgment » (FBACK). Ce message confirme si le NCoA peut être utilisé sur le nouveau lien. Si le MN s'est déplacé avant de recevoir le FBACK, et qu'il ne l'a pas reçu, il envoie un « Fast Neighbor Advertisement » (FNA) au NAR demandant confirmation pour l'utilisation du NCoA. Le NAR répond avec un « Router Advertisement » contenant une option « Neighbor Advertisement Acknowledge » (NAACK) pour indiquer si l'utilisation de ce NCoA est acceptable.

Le CN va rejeter les paquets envoyés avec l'adresse NCoA tant qu'une entrée en « binding cache » n'a pas été établie. Il est souhaitable de pouvoir continuer à utiliser une adresse qui existe dans la cache du CN jusqu'à ce que la nouvelle adresse puisse être mise à jour. Une telle mise à jour doit être rapidement exécutée.

La Figure 2.7 illustre l'échange de messages lors d'une relève avec FMIPv6.

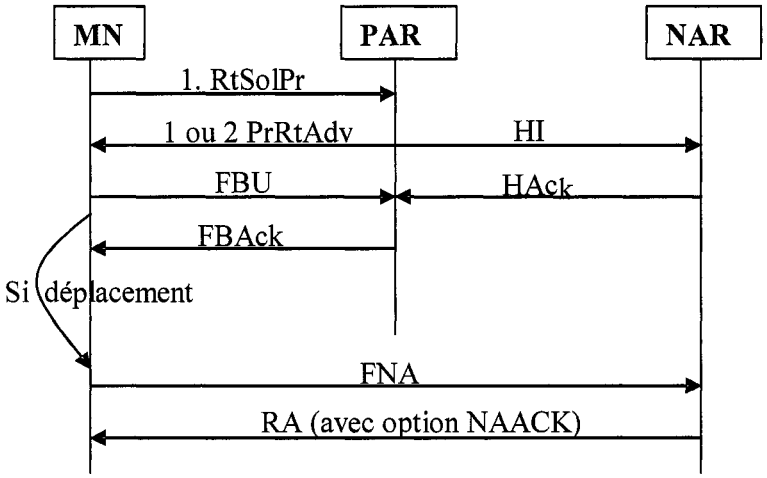


Figure 2.7 Relève rapide - échange de messages

2.1.4.4 Relève à trois participants

Le MN peut bouger d'un NAR à un autre (NAR') avant de compléter la relève et d'avoir mis à jour les associations avec ses nœuds correspondants. Le NAR devient un PAR et le NAR' un NAR. Le AAR est un AR où une association est active (il correspond à ce que l'on appelait le PAR).

Si le MN n'a pas eu le temps de configurer un CoA au PAR, le AAR est considéré comme étant le routeur par défaut quand le MN rejoint le NAR. Le MN peut transmettre un FBU au AAR pour établir un tunnel entre le AAR et le NAR. À trois participants, on enlève le tunnel entre le AAR et le PAR et on en crée un entre le AAR et le NAR. Si le MN a configuré un NCoA au PAR mais n'a pas été capable de le mettre à jour avec ses CN, le MN transmet un FBU au AAR. Le MN sera capable de recevoir les paquets arrivant au ACoA et au NCoA à travers différents

tunnels. Si le MN retourne au AAR sans avoir complété les mises à jour, il envoie un FBU avec une durée de vie de zéro pour que le AAR désactive les tunnels en sortie.

2.1.4.5 Optimisation utilisant les dispositifs de la couche liaison

Cette section décrit comment le tunnel est établi entre les AR sans messages de signalisation sur l'interface air quand la couche liaison fournit des fonctionnalités de messages d'initiation de relève (ce qui n'est pas le cas de tous les réseaux d'accès).

On appelle L2-ST le « source trigger » au PAR, L2-TT le « Target Trigger » au NAR, L2-LD le « Link Down Trigger » au PAR indiquant qu'une liaison avec un MN est terminée et L2-LU « Link Up Trigger » au NAR indiquant quand le MN vient d'établir un nouveau lien avec le NAR.

Un « Source link Layer Trigger » (couche 2 à l'origine de la relève) et un « Target Link Layer Trigger » (couche 2 ou va s'effectuer la relève) peuvent être utilisés pour initier l'échange des messages HI et HAck.

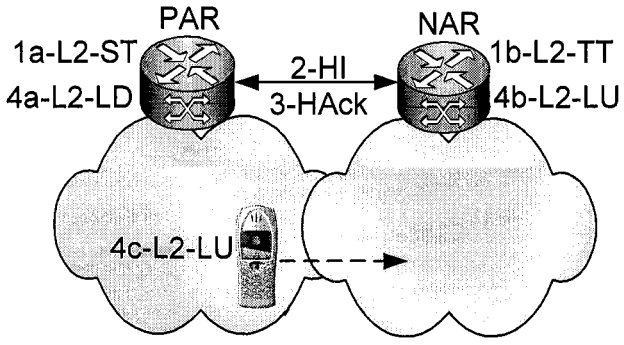


Figure 2.8 Optimisation de la relève avec informations de la couche 2

La Figure 2.8 illustre l'échange de messages lors d'une relève utilisant les dispositifs de la couche 2. Cette relève va comme suit :

1) Soit le PAR, soit le NAR reçoit une information de la couche 2 avertissant qu'un MN est sur le point de bouger. Il y a deux cas:

a) Le déclenchement L2 en est un de source (L2-ST) au PAR. Il contient l'adresse du L2 du MN et un identifiant IP pour le NAR.
b) Le déclenchement L2 en est un de cible (L2-TT) au NAR. Il contient l'adresse MAC du MN et un identifiant IP pour le PAR.

2) Le AR recevant le déclenchement doit envoyer le HI à l'autre AR. Il y a deux cas:
   a) Si le PAR envoie le HI, celui-ci contient une durée de vie, une option « IP Address » contenant le PCoA du MN et une option « LLA » (« Link Layer Address) contenant l'adresse MAC du MN. La durée de vie indique la durée de vie du tunnel.
   b) Si le NAR envoie le HI, celui-ci contient une durée de vie et une option « LLA » (adresse MAC du MN). La durée de vie contient une demande pour connaître la durée de vie dont le PAR à besoin pour le tunnel.
3) Le AR recevant le HI doit envoyer un HAck à l'autre AR. Il y a deux cas :
   a) Si le PAR envoie le HAck, celui-ci doit contenir une durée de vie, le PCoA du MN et une option « LLA ».
   b) Le NAR envoie le HAck.
4) Le début de la relève est signalé par un déclenchement L2-LD (lien tombe) au PAR. La fin est signalée par un déclenchement L2-LU (lien monte) au NAR. Les éléments impliqués manipulent le déclenchement de la façon suivante:
   a) Quand le PAR reçoit le déclenchement L2-LD, il commence à transmettre les paquets sur le tunnel vers le NAR
   b) Quand le NAR reçoit le déclanchement L2-LU, il commence à délivrer les paquets au MN et doit transmettre les paquets sortant du MN vers le PAR.
   c) Suivant l'établissement du nouveau lien, le MN configure le NCoA. Il peut différer l'obtention du NCoA mais, il ne faut pas attendre trop longtemps pour éviter que la durée de vie associée au tunnel n'expire.
5) Le PAR devient un AAR.
6) Si la relève L2 échoue à l'étape 4 ou qu'une situation de ping-pong survient, le PAR arrête d'utiliser le tunnel en envoyant un HI au NAR avec une durée de vie à zéro. Le PAR agit comme s'il n'y avait pas de relève.

Le tunnel entre les AR prend fin quand la durée de vie expire. Cependant, on peut étendre cette durée de vie.

2.1.4.6 Renouvellement de la durée de vie du tunnel

Pour étendre la durée de vie d'un tunnel, le NAR doit envoyer un HI avec une nouvelle proposition de durée de vie au PAR. Le PAR doit répondre avec un HAck indiquant si le tunnel peut être étendu. Si ce n'est pas possible, le NAR envoie au MN un RA non sollicité. Si le tunnel doit être terminé et que le MN n'a pas configurer le NCoA, il devrait commencer la configuration du NCoA sur réception du RA, au risque de perdre le service. Pour terminer le tunnel, le PAR ou le NAR doit envoyer un HI avec une durée de vie à zéro.

2.1.4.7 Relève à trois participants avec support de la couche 2

L'idée générale est que le PAR donne au NAR les mêmes informations qu'il aurait obtenu via le L2-TT. L'opération a pour but d'enlever le tunnel entre le AAR et le PAR et d'en installer un entre le AAR et le NAR. Le NAR communique avec le AAR pour déplacer le tunnel vers le NAR. Quand la relève L2 est complétée, le PAR envoie un HI au AAR pour terminer le tunnel. La Figure 2.9 décrit la séquence :

1) Le PAR ou le NAR reçoit un déclanchement de la couche 2.
   a) le déclenchement vient da la source (L2-ST). Il contient l'adresse MAC du MN et un identifiant de l'adresse IP ou l'adresse IP du NAR.
   b) Le déclenchement vient de la destination (L2-TT). Cet événement contient l'adresse MAC du MN et un identifiant de l'adresse IP du PAR.
2) Le PAR et le NAR échangent des HTT/HI ou des Hack/HTT.
   a) Le déclenchement vient de L2-ST. Le PAR doit envoyer un message relève à trois (HTT : « *Handover to third* ») au NAR contenant l'adresse IP du AAR et deux LLA. Le premier LLA doit contenir l'adresse MAC du MN et le second l'adresse MAC du AAR. Le NAR doit répondre avec un HAck.

b) Le déclenchement vient de L2-TT. Le NAR doit envoyer un HI au PAR comme s'il s'agissait d'une relève à deux participants. Le PAR doit répondre avec un HTT contenant l'adresse IP du AAR et les deux LLA.

3) Le NAR doit d'abord vérifier ses tables de routage pour voir si un tunnel est déjà là pour le MN. Si oui, on saute à l'étape 6. Si non, le NAR doit s'occuper de la relève avec le AAR et échanger des messages Hi/HAck. Comme le AAR ne reçoit aucune indication de déclenchement de la part de la couche 2 lui disant quand commencer la relève, il peut placer un tunnel bidirectionnel vers le NAR après avoir transmis le HAck ou alors, il doit attendre d'établir le tunnel avec le NAR tant qu'il n'a pas reçu du PAR un message indiquant que la relève L2 (L2-LD) a commencé (ce qui arrive a l'étape 4).
4) Au départ de la relève L2, le AAR et le PAR échangent des messages pour terminer le tunnel qu'il y a entre eux deux et pour permettre au AAR de commencer l'établissement du tunnel avec le NAR.
   a) Le début de la relève L2 est signalé au PAR par L2-LD.
   b) Le PAR doit échanger des messages HI/HAck avec le AAR avec une valeur de durée de vie à zéro. Cela termine le tunnel entre les deux. Si le AAR n'a pas placé de tunnel avec le NAR, il doit le faire sur réception du HI.
5) La fin de la relève L2 est signalée par L2-LU. On le traite de la façon suivante :
   a) Le NAR doit commencer à transmettre les paquets au MN.
   b) Suivant son mouvement et son flot de trafic, le MN peut chercher à obtenir un NCoA.
6) Dans un cas spécial où le NAR et le AAR sont les mêmes, le AAR reconnaît qu'il a un tunnel avec le PAR. Il s'en rend compte à l'étape 3. Dans ce cas, le AAR n'a pas à s'envoyer les HI/HAck (de l'étape 3). Sur réception du L2-LU (fin de la relève), le AAR doit router les paquets directement au MN. Le AAR échange les HI/HAck (étape 4) avec le PAR car celui-ci ne peut pas savoir que le AAR et le NAR sont les mêmes.

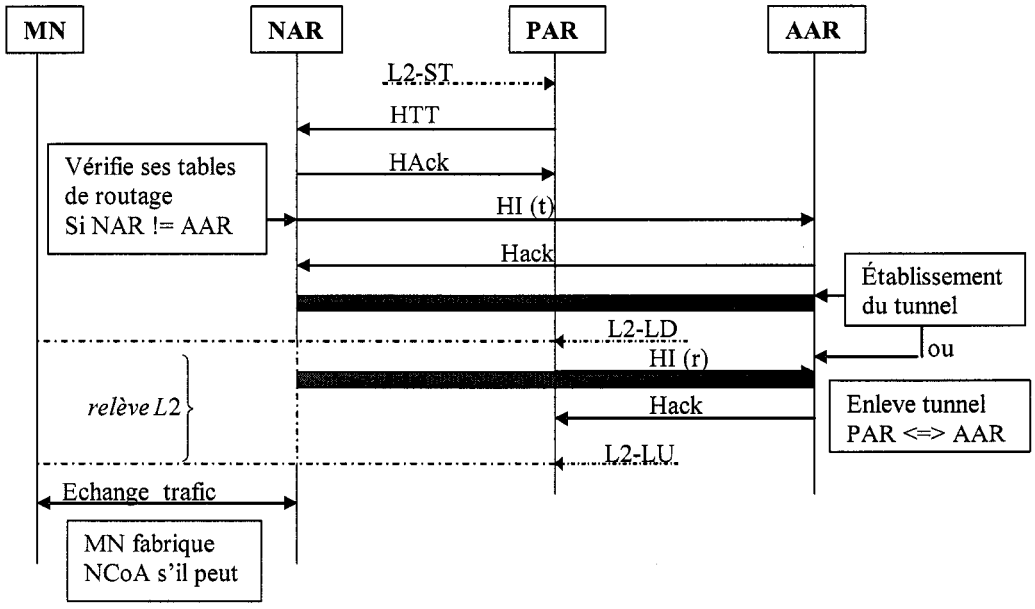


Figure 2.9 Relève à trois participants initié au PAR

FMIPv6 permet l'anticipation de la relève et redirige le trafic vers la nouvelle position du MN avant qu'il ne s'y déplace. Cependant, il n'est pas aisé de déterminer le temps où commencer la redirection du trafic entre le PAR et le NAR. Des paquets sont perdus si c'est fait trop tôt ou trop tard. De plus, des corrections sont nécessaires pour le cas ou le MN va et vient très vite entre les AR (ping-pong). Il est nécessaire de découpler la synchronisation de la relève L3 de la synchronisation de la relève L2. Une solution est le « bicasting » ou le « n-casting » [5] de paquets destinés au MN, pour une petite période de temps, du vieux point d'attache vers un ou plusieurs emplacements futurs potentiels avant que le MN n'y arrive.

2.1.5 Le Bi-casting

FMIPv6 décrit comment minimiser l'interruption de service durant la relève. Le « Simultaneous Bindings » [5] (bi-casting ou n-casting) est une extension à FMIPv6 avec des fonctions d'associations simultanées pour minimiser la perte de paquets. Le trafic vers le MN est « bicast » ou « n-cast » pour une petite période à son endroit courant et à un ou plusieurs endroits où on s'attend à retrouver le MN.

Cela enlève l'ambiguïté sur le moment où commencer à envoyer le trafic pour le MN à son nouveau point d'attache et enlève l'interruption lors de mouvements de ping-pong. La procédure de relève décrite précédemment est améliorée en envoyant des paquets au MN par plusieurs AR.

2.1.5.1 Opération du MN

Un MN avec une association active recevant un RA (PrRtAdv) doit établir de nouvelles associations. Il doit envoyer un FBU avec le drapeau association simultanée à 1. Deux valeurs de durée de vie sont retournées :

1. «Bicating lifetime » : durée de vie du « bicasting » ;
2. la durée de vie du nouveau CoA.

La durée de vie du nouveau CoA fonctionne parallèlement à la durée de vie de l'association simultanée. Quand la durée vie du « bicasting » finit, le MN enlève cette entrée et garde une entrée pour le nouveau CoA avec la durée de vie restante.

2.1.5.2 Opérations HA/MAP/AR

Le HA, le AR et le MAP sont ceux qui peuvent recevoir les FBU. Sur réception du FBU avec le drapeau d'association simultanée à 1, le HA, le MAP ou le AR doivent créer une nouvelle sous-entrée pour le nouveau CoA relié à l'entrée originelle du vieux CoA. Cette sous-entrée contient les mêmes champs qu'une entrée normale mais possède deux durées de vie :

1. la durée de vie normale du nouveau CoA,
2. la durée de vie de l'association simultanée.

Les deux durées de vie roulent en parallèle. Jusqu'à ce que la durée de vie de l'association n'expire, le HA, le MAP ou le AR doivent envoyer une copie des données destinées au MN aux deux adresses CoA. Quand la durée de vie de l'association expire, on remplace l'ancienne entrée par la nouvelle.

2.1.5.3 Copies multiples des paquets

Si le MN a des associations simultanées avec le HA/MAP ou AR, il peut recevoir plusieurs copies du même trafic. L'utilisation des associations simultanées ne signifie pas obligatoirement que le MN recevra les paquets de plusieurs sources. La Figure 2.10 illustre le flot de données arrivant au MN lors d'un « bicasting ».

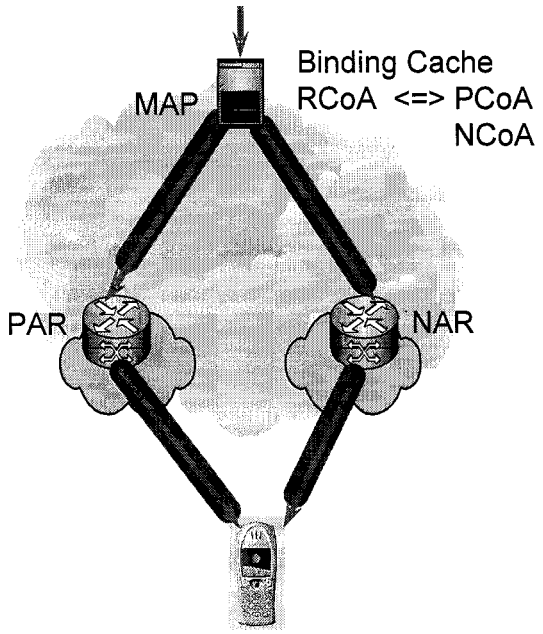


Figure 2.10 Bi-casting - Flot de données

La gestion de la mobilité comprend les différents protocoles et concepts vus dans la section 2.1. Cependant, le but de ce mémoire est de définir un modèle d'API pour la gestion de la mobilité dans un système bien précis. La section suivante à pour but d'étudier le système où sera implanté cet API.

2.2 Architecture générale du système

Les éléments de réseaux comme les commutateurs et les routeurs peuvent être divisés en trois composantes logiques opérationnelles, comme décrit dans le groupe de travail ForCES [12] :

1. le niveau contrôle (CE : « *Control Element* ») : contrôle et configure le niveau acheminement;

2. le niveau acheminement (FE : « *Forwarding Element* ») : manipule le trafic du réseau;
3. le niveau gestion : s'occupe de la gestion des deux niveaux précédents.

En général, le CE exécute différents protocoles de signalisation et de routage et fournit toutes les informations de routage au FE. Le CE contrôle un FE en manipulant ses tables de routage ou l'état de ses interfaces. Le FE prend les décisions suivant les informations reçues et s'occupe des opérations sur les paquets telles la transmission, la classification, le filtrage... Par exemple, dans un routeur, le CE exécute les protocoles de routage, le FE exécute une commutation matérielle et le niveau gestion arrête ou commence les processus de routage. Un ensemble de mécanismes pour relier ces composants fournit une évolutivité accrue et permet aux deux éléments d'évoluer indépendamment. Le CE est plutôt de type logiciel car il doit supporter un grand nombre de protocoles et de fonctionnalités. Si un changement arrive dans les requis d'un protocole, il est donc assez aisé de le mettre à jour. Le FE est de type matériel. Le FE est fait de processeur réseau (NP : « Network Processor »). Les NP évoluent très rapidement de sorte que l'on peut améliorer les performances d'un routeur juste en changeant une carte FE. Cela peut se faire sans changement au niveau du CE ou du comportement de la machine. En permettant aux CE et FE d'évoluer indépendamment, différents types de FE peuvent être développés, certains d'usage universel et d'autres plus spécialisés.

La standardisation d'APIs permet de réunir et de faire fonctionner des composantes hétérogènes. Les APIs du « Network Processing Forun » (NPF) sont conçus pour cela car il présente des interfaces de programmation flexibles. Ils permettent l'existence de plusieurs FE de manière transparente. De plus, les propriétés matérielles et la nature des interconnexions entre les niveaux contrôle et acheminement sont indépendantes.

Plusieurs impératifs architecturaux sont requis [10] :

1. les différents niveaux doivent pouvoir s'interconnecter par un ensemble varié de technologies ;
2. les FE doivent supporter un ensemble minimal de fonctionnalités nécessaires pour établir la connectivité de réseau ;

3. les paquets doivent pouvoir arriver au NE (« Network Element ») par un FE et le quitter par l'intermédiaire d'un autre ;
4. un NE doit avoir l'apparence d'un seul élément. Par exemple, le champ TTL d'un paquet devrait être décrémenté seulement une fois pendant qu'il traverse le NE, indépendamment du nombre de FE par lequel il passe ;
5. l'architecture doit fournir une manière d'empêcher les éléments non autorisés de joindre un NE ;
6. un FE doit pouvoir informer le niveau contrôle de l'augmentation ou de la diminution des ressources. Par exemple, le nombre de ports physiques ou la quantité de mémoire assignée peut changer et le CE doit en être informé ;
7. l'architecture doit supporter des mécanismes de redondance ;
8. les CE doivent pouvoir réorienter des paquets de commande adressés à leurs interfaces de contrôle. Les CE doivent pouvoir configurer la redirection de paquet sur les FE ;
9. le FE doit fournir ses informations de topologie au CE ;
10. l'architecture doit supporter des niveaux de contrôle et d'acheminement multiples.

La variété de fonctionnalités du FE pose un problème potentiel pour le CE. Pour qu'un CE gère efficacement un FE, le CE doit comprendre comment le FE traite les paquets. Nous avons donc besoin que le modèle FE exprime les possibilités de traitement du paquet. Le FE possède certains requis en terme de fonctionnalité et doit pouvoir informer le CE et lui fournir:

- le nombre de ports et leurs attributs ;
- données relatives à la transmission de paquets ;
- les capacités en terme de QoS, de filtrage et de classification ;
- il doit pouvoir supporter l'ajout de fonctionnalités ;
- il doit indiquer le type de mécanismes et de traitement de paquets.

Les FE ne prennent aucune initiative. Ils sont esclaves. Il appartient au CE de les coordonner pour réaliser la redondance, le partage de charge ou des commandes

distribuées. L'idée est de maintenir les FE aussi simples que possible de sorte qu'ils puissent concentrer leurs ressources sur les fonctions de traitement de paquets.

2.2.1 Module du niveau contrôle

Les caractéristiques du niveau contrôle sont les suivantes :

1. Un CE doit être capable de travailler avec plusieurs FE et posséder l'intelligence suffisante pour faire du « mapping » de un à plusieurs. Par exemple, pour chaque nouvelle route, un CE doit être capable de la transmettre aux différents FE présents.
2. Il se rend compte de la topologie des FE.
3. Association et capacité de découverte quand le CE détecte des nouveaux éléments d'expédition.
4. Un plugiciel s'occupe des mécanismes de transport et de protocole pour la communication avec les FE.
5. Il implémente différents API (expédition, namespace, configuration et gestion, classification, MPLS, DiffServ...)
6. Il utilise une couche d'abstraction d'OS pour l'indépendance entre le matériel du niveau contrôle et son OS.

Le NPF définit deux ensembles d'API : l'API d'application qui est une collection de sous-API servant à fournir des fonctionnalités spécifiques (IPv4 Unicast Forwarding, MPLS...) et l'API de gestion d'interface. Cet API définit la relation entre les interfaces de la couche 3 et les tables de routage locales. Les API sont soumises à des efforts de standardisation à l'intérieur du NPF. Comme le processus de standardisation est en évolution constante, les implémentations peuvent requérir quelques modifications.

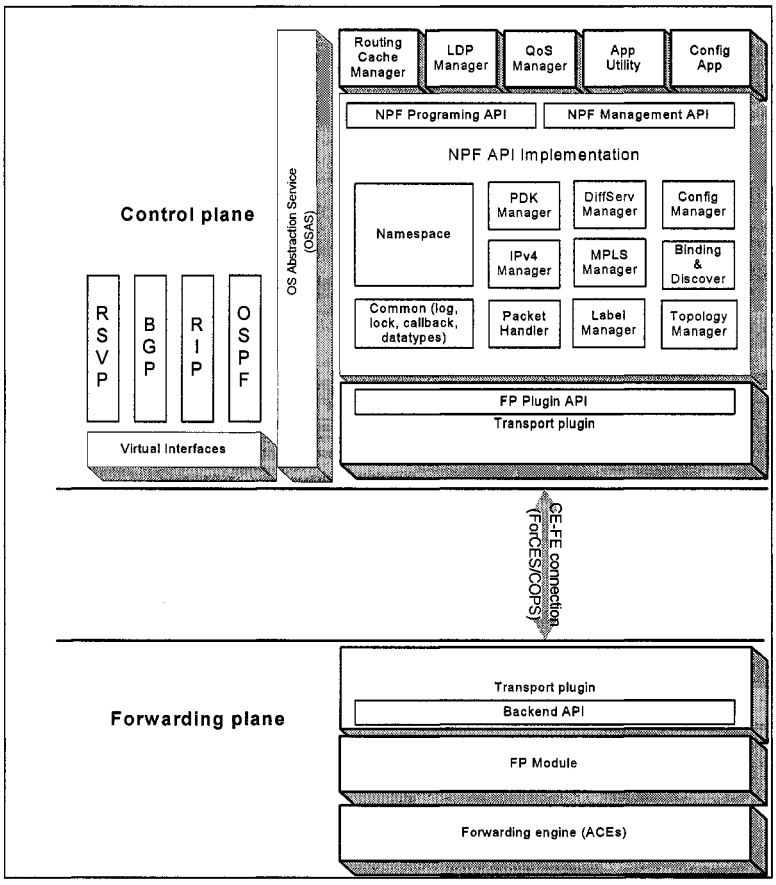


Figure 2.11 Architecture du système

La Figure 2.11 montre l'architecture du système [13]. On peut voir que le CE est composé des modules suivants :

1. Module d'implémentation de l'API d'application (e.g. IPv4 Unicast Forwarding) ;
2. Module de configuration et de gestion ;
3. Module « Namespace » ;
4. Module d'association et de capacité de découverte ;
5. Gestionnaire de la topologie du niveau acheminement ;
6. Module de transmission inter-niveaux acheminement ;
7. Identificateur de Callback et d'évènements ;
8. Couche d'abstraction d'OS ;
9. Gestionnaire du niveau contrôle ;

10. Module multi client du niveau contrôle ;
11. Module de services et de protocoles fournissant le support pour les applications et le routage. Il consiste en :
    a) un pilote d'interface virtuelle qui fournit une abstraction pour des interfaces multiples de FE comme des interfaces virtuelle dans le niveau contrôle ;
    b) un identificateur de paquet de l'élément de contrôle ;
    c) un « Route Cache Manager » (RCM) : agit comme proxy pour le gestionnaire de la table de routage et permet de fonctionner sans modification spécifique du CE.

2.2.1.1 Application API Implémentation Module

Ce module implémente les applications des API fournissant le support au développement des applications. Ils masquent l'existence de plusieurs niveaux acheminement et en exposent les détails si les applications de contrôle en ont besoin. Ceci peut se faire grâce au Namespace. Par exemple, l'API IPv6 unicast forwarding fournit des objets pour la configuration et la gestion des tables de routage IP et ARP [14].

2.2.1.2 Module de configuration et de gestion

Ce module implémente les méthodes définies par l'API de configuration et de gestion. Il est utilisé par les applications pour configurer et gérer les éléments du réseau. Ces API facilitent la configuration et la gestion des interfaces, des ports, des tables de routage IP, etc.

2.2.1.3 Module Namespace

Ce module implémente les « Namespaces » qui sont utilisés par les applications de gestion multiple pour identifier, localiser et grouper les objets à gérer dans le système. Cet API contient tous les objets configurables du système. Cela

inclut des objets représentant des entités individuelles sur différents niveaux d'acheminement comme les interfaces physiques et les tables ARP [14]. Il y a aussi des objets qui représentent, par exemple, toutes les tables de tous les niveaux acheminement.

2.2.1.4 Module d'association et de capacité de découverte

Ce module est responsable de l'association et de la découverte des FEs et s'occupe aussi de leurs initialisations. Ce module appelle le module de configuration et de gestion pour ensuite peupler les Namespaces suivant les informations apprises.

2.2.1.5 Gestionnaire topologique du niveau d'acheminement

La manière dont une opération particulière est faite (e.g. mise à jour d'une route) dépend de la topologie de connexion des FEs. Les FE peuvent être connectés en bus, en étoile ou autre, ce qui affecte les opérations de contrôle et les données transportées. Ces données peuvent devoir être modifiées dans certains cas pour simuler le comportement d'un routeur virtuel, ce sous-module en est responsable.

2.2.1.6 Module d'identificateur de callback et d'évènement

Les API sont asynchrones. Les applications utilisant les registres de « callback » sont appelées après qu'on ait complété un appel. Les évènements sont aussi reportés à l'application à travers des callbacks. Le « callback » est une technique de programmation où un processus en déclenche un autre. Le deuxième appelle plus tard le premier comme résultat d'une certaine valeur, d'une action ou d'un événement.

2.2.1.7 Module de gestion du niveau contrôle

Ce module initialise le CE et fournit une infrastructure robuste pour les différentes applications en les empêchant d'entrer dans un état de blocage.

2.2.1.8 Service de support de protocole

Pour utiliser les protocoles de routage et les applications de contrôle sans modifications, ont doit fournir un support additionnel. Les protocoles de routage existants et les applications de contrôle utilisent une interface pour envoyer et recevoir les paquets de données. Les modules suivants permettent de compléter le support pour les applications :

1. Pilote d'interface virtuel (VIDD : « Virtual Interface Device Driver ») : permet aux applications, aux interfaces de transmission et aux protocoles de routage de voir l'interface du FE comme une interface virtuelle sur le CE ;
2. Identificateur de paquet pour le FE : les paquets destinés pour les protocoles et applications sur le CE doivent venir du FE. De la même manière, des paquets du CE transmis aux interfaces virtuelles doivent être transportés au FE. Ces paquets sont transportés en utilisant le protocole ForCES ;
3. Gestionnaire de la mémoire cache de routage : Un mécanisme doit pouvoir fournir le maintien de la synchronisation de la table de routage du CE avec les tables de routage de chaque FE.

2.2.2 Transport entre niveaux contrôle et acheminement

Les CE et FE utilisent différents mécanismes pour l'échange d'information. Cela peut être soit des protocoles standards de l'IETF (ForCES, COPS [15] ou GSMP [16]), soit des mécanismes comme CORBA [17]. Les différents niveaux peuvent être connectés en utilisant différents types d'interconnexion (PCI, InfiniBand, switch fabric, mémoire partagée...). La couche transport fait abstraction du type et du détail des mécanismes de communication pour fournir des fonctionnalités plug-and-play aux communications. Le CE appelle l'API du FP Plugin et l'agent ForCES translate ces appels en messages ForCES. Ces messages sont envoyés à l'agent du FE qui parse les messages et appelle les fonctions du FP module à travers l'API Back-end.

2.2.3 Niveau acheminement

Le FP module est un module important de ce niveau (Figure 2.11). Il spécifie le FPPAPI qui est utilisé comme interface de programmation. Ce module reçoit des appels API et fait la correspondance avec les appels de fonctions. Il est aussi responsable du support d'association et de découverte des FE et des communications entres les différents niveaux. L'API Back-end fait la transition entre le niveau transport et le module d'acheminement, s'occupe de la gestion du FE (ajouter/enlever des routes...) et fournit les supports suivants :

1. association ;
2. niveau applicatif du API ;
3. notification d'évènements (e.g. interface tombe ou monte).

Maintenant que nous avons analysé la gestion de la mobilité et l'architecture du système où va être implémenté l'API de gestion de mobilité, nous pouvons voir la conception de cet API.

CHAPITRE 3

API POUR LA GESTION DE LA MOBILITE

Un *API* est une interface de programmation d'applications, un jeu de fonctions utilisé pour accéder à certaines fonctionnalités. Il comprend tout ce qui relie un programme avec son environnement ainsi que tous les utilitaires, c'est-à-dire les structures de données fréquemment utilisées. L'API a pour but d'uniformiser les applications car toutes les structures de données et tous les appels de fonction seront les mêmes, indépendamment de la plate-forme sur laquelle on l'implémente et quelle que soit la manière dont on implémente les fonctions. L'API est strictement la même sur toutes les plates-formes (Windows, Unix, Linux). Dans ce chapitre, nous proposerons de concevoir un API pour la gestion de la mobilité. Cette conception se base sur ce que nous avons vu dans le chapitre précédent mais doit également tenir compte de ce qui a déjà été fait pour IPv6. Nous n'avons pas à redéfinir IPv6. Elle nécessite en effet que l'on décrive les différentes tables utilisées, propose une séparation entre les niveaux *contrôle* et *acheminement* ainsi que les différentes fonctions et structures associées. Deux API vont donc être définis : l'API de gestion de mobilité qui définit un ensemble de structures et de fonctions de manière à maintenir l'ensemble des informations contenues au niveau du FE et l'API de gestion d'interface qui définit la manière selon laquelle les interfaces de la couche 3 sont associées aux tables locales de routage du FE (FIB : « *Forwarding Information Base* ») ainsi que la manière selon laquelle les adresses IP sont associées à une interface de la couche 3.

3.1 Détails d'environnement

Il y a au moins deux bases de données requises pour l'expédition de paquets IP [18]. La première est la base d'information de routage (RIB : *Routing information base*), qui réside sur le CE, et la seconde est la base d'information de transmission (FIB : *Forwarding information base*) qui se trouve sur le FE et qui est donc accessible par les processeurs réseau.

Les paquets en entrée ont leur adresse de destination extraite et elle est utilisée comme référence pour la consultation de la FIB. Si une correspondance est trouvée, le FIB fournit l'adresse IP du prochain saut et l'interface de sortie à utiliser pour faire le saut.

Le RIB peut être défini par configuration statique ou par l'intermédiaire des protocoles de routage dynamique comme OSPF [19] ou BGP [20]. Un tel logiciel de couche application se connectera par une interface à un composant du gestionnaire de la table de routage (RTM : *Route Table Manager*). Le RTM gère le RIB et maintient le FIB. Habituellement, le RIB contient toutes les routes connues de tous les protocoles de routage, et le FIB est l'élément actif choisissant le meilleur chemin. Le RTM utilise les appels de fonction définis par l'API pour gérer un FIB. Un exemple de système pourrait avoir les caractéristiques suivantes :

- Un RTM gérant un RIB au niveau du CP utilise des appels de service API pour maintenir un FIB à l'usage d'un processeur réseau (NP : *Network Processor*) ;
- Un NP sur le FE a les connaissances et contrôle une ou plusieurs interfaces de la couche 3 ;
- Le NP a la connaissance et l'accès au FIB ;
- Chaque FIB est associé à une ou plusieurs interfaces de la couche 3 ;
- Le FIB est mis en référence par un identifiant unique ;
- Un ou plusieurs NP peut être présent dans le système.

À la Figure 3.1, le RTM surveille la gestion des routes apprises grâce aux protocoles de routage et maintient un RIB. En utilisant l'API, le RTM définit et peuple le FIB. Le CP a la connaissance sur un seul FIB. Un FIB particulier peut être

répliqué dans différents NP. Une autre implémentation peut représenter un système qui a créé des routeurs virtuels multiples afin de réaliser un réseau privé virtuel. L'isolement est fourni entre les domaines de routage en maintenant les RIB indépendants, et par conséquent des instances uniques de leurs FIB associés. Dans cette situation, le CP a la connaissance de plusieurs FIB.

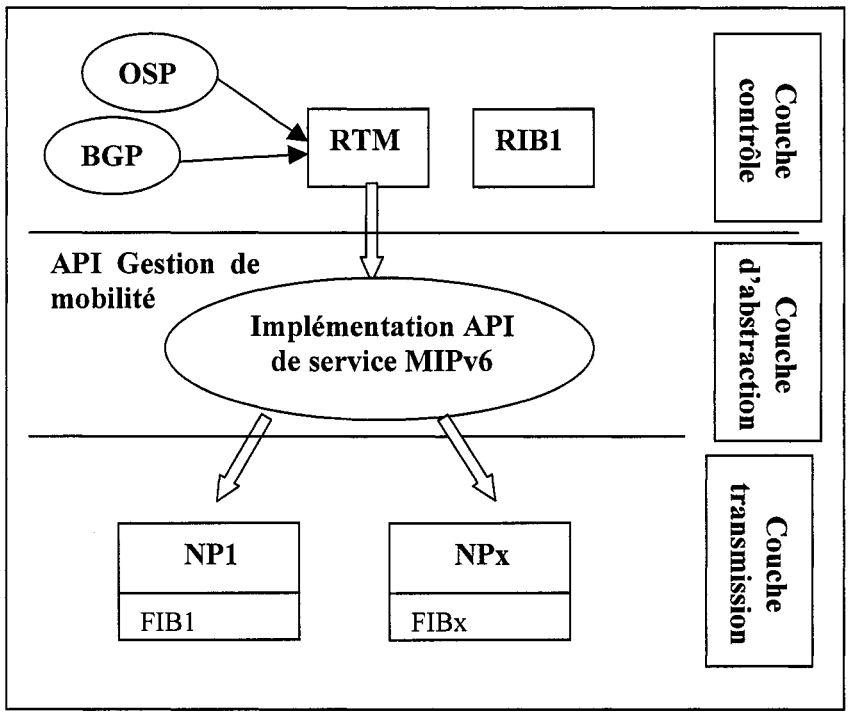


Figure 3.1 Exemple de système

3.1.1 Modèle d'utilisation

L'architecture des systèmes et le design des applications permettent de modéliser un FIB de plusieurs manières. Avec IPv6, deux modes pour organiser et manipuler l'information au niveau du service API ont été développés. Le premier mode, appelé *mode de table unifié*, utilise une seule table pour structurer et gérer l'information de transmission. Le deuxième mode, appelé mode *de table discrète*, utilise des tables de préfixe et de prochain saut séparées pour conserver l'information de transmission. Les deux modes représentent l'information concernant la résolution

d'adresse en utilisant une table séparée. Les modes n'impliquent pas que les FE soutiennent l'un ou l'autre de ces modes. L'application n'a aucune connaissance des différents éléments d'expédition. Par exemple, un élément d'expédition peut implémenter les tables de préfixe et de prochain saut en mode discret, tandis que l'application de routage du CP peut organiser son information FIB en mode unifié. L'exécution des services API sera responsable de mapper les paramètres unifiés à un format approprié pour l'organisation en mode discret du FE.

Les implémentations adoptant le mode de table unifié utilise une seule entité de données, qui doit faire référence à la table FIB (Table FIB = mode unifié de FIB). Cette table est composée d'entrées et chacune d'entre elles se compose d'un préfixe et d'un tableau de prochain saut. Pour faciliter l'équilibrage de charge, le tableau de prochain saut peut contenir l'information pour un ou plusieurs sauts. La Figure 3.2 illustre la disposition d'une table FIB. Dans cette structure, on référence une entrée par le préfixe.

Les implémentations qui adoptent le mode de table discret utilisent des tables de données séparées. La table de préfixe est composée d'entrées, chacune se compose d'un préfixe et d'un identificateur de prochain saut qui indique une entrée dans une table de prochain saut. La table de prochain saut est composée d'entrées, chacune se compose d'un identificateur de prochain saut et d'un tableau de prochain saut. Comme avec la table en mode unifié, le tableau de prochain saut peut contenir une ou plusieurs entrées de prochain saut.

Afin de transmettre un paquet, chaque adresse de destination IP spécifiée dans le préfixe doit être reliée à un ou plusieurs prochains sauts. En mode discret, cette association est fournie par l'identificateur de prochain saut, qui associe une entrée de table de préfixe avec une entrée dans la table de prochain saut. Ce modèle de table séparée présente plusieurs avantages dans la conception des systèmes. Par exemple, quelques classes de nœuds du réseau requérant une performance élevée exige des mises à jour rapides de FIB quand un ensemble de routes changent. Avec une implémentation en mode discret, il peut être possible de mettre à jour efficacement l'information de routage en changeant un sous-ensemble d'entrées de la table de prochain saut. Un mode unifié exige qu'un plus grand ensemble d'entrées de table soit modifié pour accomplir la même mise à jour de l'information de routage.

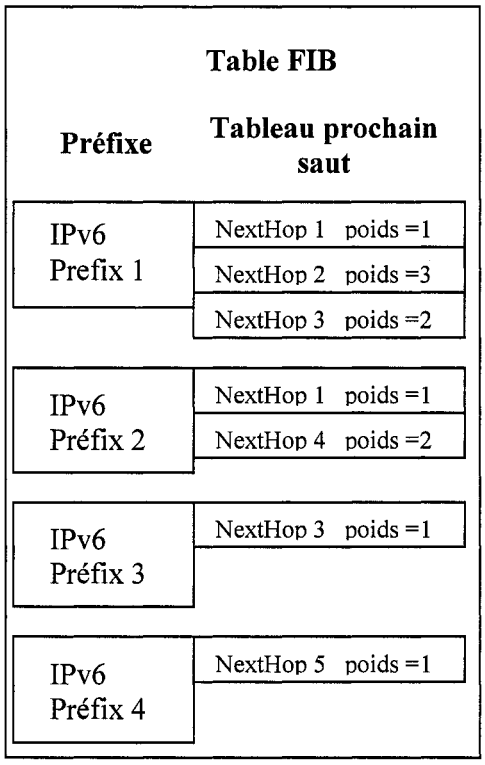


Figure 3.2 Table FIB en mode unifié

La Figure 3.3 illustre la disposition et le rapport entre la table de préfixe et la table de prochain saut. Dans la table de préfixe, la clef utilisée pour référencer une entrée est le préfixe. Dans la table de prochain saut, la clef de référence d'une entrée est l'identificateur de prochain saut.

Une application peut créer plusieurs tables de préfixe et de prochain saut. L'API définit une fonction qui crée une relation entre les tables. La relation peut être de un à un, de sorte qu'il y ait une table de préfixe correspondant à chaque table de prochain saut. Elle peut être de plusieurs à un dans ce cas, plusieurs tables de préfixe partagent une seule table de prochain saut. La relation de un à plusieurs où une seule table de préfixe est associée à plusieurs tables de prochain saut n'est pas supportée.

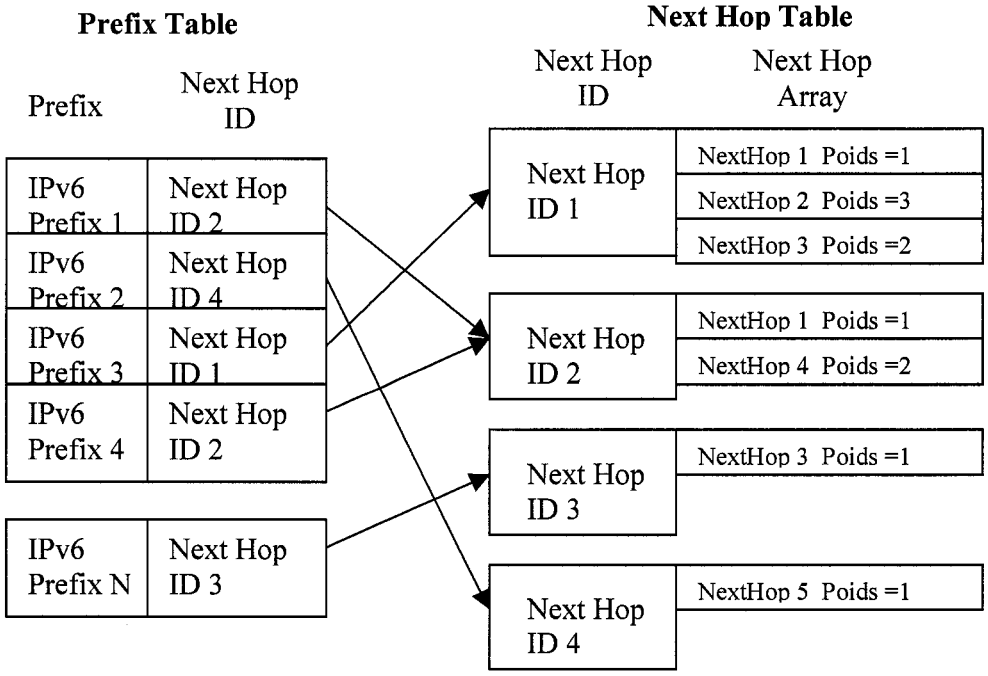


Figure 3.3 Table FIB en mode discret

Les applications clients des services API créeront une ou plusieurs instances des tables. Afin de déterminer quel type de table créer, on peut utiliser les méthodes *Query* pour déterminer quels modes de table sont supportés et quel mode fournit la meilleure performance. Une fois qu'une application a déterminé le mode de fonctionnement, elle peut créer les tables en utilisant les fonctions liées à chaque type de table et ensuite remplir et surveiller chaque table.

3.1.2 Application à MIPv6

Avec MIPv6, nous avons défini plusieurs tables ou fait des ajouts aux tables existantes [21]. Ainsi, on retrouve :

1. une liste de destinataires,
2. une liste de routeurs par défaut,
3. une liste des associations d'adresse,
4. une liste de HA,

5. une liste de MAP,
6. une liste pour la gestion des tunnels (HA⇔MN et MAP⇔MN),
7. une liste de préfixe agrégé et
8. une liste de CoA (« CoA pool »).

Toutes les tables vont avoir une représentation séparée, sauf la table de gestion du tunnel qui peut être en mode séparé ou unifié. Puisque chaque association d'adresse, au niveau du MAP et du HA, nécessite la création d'un tunnel vers le MN, la table d'association d'adresse possède un identifiant de la table de gestion de tunnel. Non seulement cela permet de faire une association entre l'association d'adresse et le tunnel correspondant, mais aussi cela simplifie les mises à jour.

La mise à jour de la table d'association d'adresse entraîne une mise à jour du FIB. Cependant, le FIB ne connaît pas la durée de vie de l'association. Cette durée de vie est seulement connue dans la table d'association d'adresse. Il faut donc tout le temps faire attention à l'intégrité du FIB. Celui-ci ne doit pas voir une adresse CoA valide alors que la durée de vie a expiré.

3.2 Structures de données nécessaires

Différentes structures de données sont nécessaires. On retrouve tout d'abord les structures pour l'interface avec les ports physiques et logiques, incluant les aspects de l'interconnexion avec les liens physiques ou la couche 2. Cela fournit les informations nécessaires à la configuration et à la gestion des interfaces physiques et logiques, mais non celles dont l'application a besoin. Il faut donc surtout définir les attributs des nouvelles interfaces, ainsi que toutes structures de données nécessaires au bon fonctionnement des différents protocoles et des applications.

3.2.1 Attributs d'interface

On représente une interface avec une hiérarchie de structures. Au niveau supérieur, on trouve la structure contenant les attributs qui peuvent être réglés par l'application et qui sont communs à tous les types d'interface. À l'intérieur de cette

structure, il y a un *Union* contenant les objets qui sont les attributs spécifiques à un type d'interface. En ce qui nous concerne, on utilise deux types d'interface : l'interface MIPv6 et l'interface tunnel (*inner IP*) pour les communications entre le HA ou le MAP avec le MN.

Attributs communs

La structure des attributs communs d'une interface contient les objets suivants :

1. Identificateur d'interface : valeur entière différente de zéro assignée par l'application à chaque interface qu'elle crée. Deux interfaces ne peuvent pas avoir le même identificateur ;
2. Type d'interface : code indiquant lequel des groupes d'attributs va être utilisé (IPv4, ATM, Tunnel MIPv6…) ;
3. Statut administratif : contrôle général de la mise en fonction ou non de l'interface ;
4. Vitesse du lien pour l'interface.

Attributs interface MIPv6

Cette interface représente essentiellement une association entre les attributs MIPv6 et le port logique ou physique [22]. Ces attributs sont :

- les adresses MIPv6 ainsi que leurs préfixes,
- leur durée de vie,
- les adresses de réception multicast,
- le MTU, et
- un identificateur de la table des informations de transmission (FIB).

Attributs interface de tunnel

Le *tunneling* est une technique consistant en l'établissement et en l'utilisation d'un lien virtuel entre deux nœuds pour transmettre les paquets. Du point de vue des nœuds, ces liens virtuels apparaissent comme étant un lien point à point sur lequel le protocole de tunnel agit comme un protocole de couche 2.

Un paquet du tunnel IPv6 est limité à la taille maximale du paquet. Quand on a des tunnels dans un tunnel, on ajoute une nouvelle encapsulation (IPv6 dans ce cas ci), ce qui augmente la taille du paquet. En conséquence, le nombre d'entêtes de tunnel, et donc le nombre d'encapsulations internes est limité par la taille maximale du paquet. Comme l'augmentation de la taille du paquet peut exiger la fragmentation à un point d'entrée du tunnel, la limitation des encapsulations est recommandée.

Un tunnel est modélisé comme étant un lien virtuel d'un seul saut. Dans le cas d'un chemin d'expédition avec plusieurs niveaux de tunnels, une boucle dans le cheminement d'un tunnel interne à un tunnel externe est particulièrement dangereuse quand les paquets des tunnels internes ré-entrent dans un tunnel externe dont ils ne sont pas encore sortis. Dans un tel cas, l'encapsulation interne devient une encapsulation récursive. Puisque chaque encapsulation ajoute une entête de tunnel avec une nouvelle valeur limite sur le nombre de prochains sauts, le mécanisme de limite de saut ne peut pas contrôler le nombre de fois où le paquet atteint le nœud externe d'entrée du tunnel, et ne peut pas contrôler le nombre d'encapsulations récursives. Le nombre maximum de sauts que le paquet peut traverser doit être contrôlé par deux mécanismes utilisés ensemble pour éviter les effets négatifs de l'encapsulation récursive dans des boucles de cheminement:

- la limite de nombre de sauts du paquet : elle est décrémentée à chaque saut du paquet original ;
- la limite d'encapsulation de paquet de tunnel : elle est décrémentée à chaque encapsulation interne du paquet.

Si le protocole de tunnel en est un de couche 3, alors l'interface de tunnel doit être un enfant d'une interface de la couche 3 (IPv6). Le tunnel est utilisé entre le HA et le MN et/ou entre le MAP et le MN. Les attributs de cette interface sont :

- Protocole du tunnel (IPv6 dans IPv6) ;
- MTU (MTU max du tunnel = (MTU IPv6 - taille entête IPv6)) : Le MTU est réglé dynamiquement suivant le *path MTU* entre le nœud d'entrée et de sortie du tunnel moins la taille des entêtes de tunnel. On doit donc faire une découverte du *path MTU* (message ICMP) entre le point d'entrée et de sortie

du tunnel : si on a plusieurs tunnels les uns dans les autres, le MTU est celui du tunnel externe moins la taille des entêtes que l'on doit rajouter ;

- Nombre de sauts limite. La limite de saut de tunnel doit être configurée avec une valeur qui assure que le paquet tunnelé puisse rejoindre sa destination (assez grand) et évite les boucles ;
- Nombre d'encapsulation de tunnel limite ;
- Label de flot ;
- Adresse du nœud d'entrée du tunnel ;
- Adresse du nœud de sortie du tunnel ;
- Identificateur de la table FIB.

3.2.2 Structures de données MIPv6

Nous allons maintenant définir l'information minimale que devrait contenir les différentes structures pour assurer le bon fonctionnement du protocole MIPv6 [21]. On retrouve principalement :

1. une liste de destinataires,
2. une liste de routeurs par défaut,
3. une liste des associations d'adresse,
4. une liste de HA,
5. une liste de MAP,
6. une liste pour la gestion des tunnels (HA⇔MN et MAP⇔MN),
7. une liste de préfixes agrégés et
8. une liste de CoA (« CoA pool »).

Les structures de données des adresses IPv6 sont définies dans [23].

3.2.2.1 Liste de destinataires

La liste des destinataires conserve un ensemble d'entrées au sujet des différentes destinations où du trafic a été envoyé récemment. En ce sens, elle

ressemble énormément à la liste des voisins déjà définie avec IPv6. Elle est composée d'un ensemble d'entrées qui sont définies dans la liste suivante :

1. adresse unicast IP,
2. adresse MAC,
3. un drapeau indiquant si le voisin est un routeur ou non,
4. un pointeur à tous les paquets en queue attendant l'accomplissement de la résolution d'adresse,
5. l'état d'accessibilité :
    a. INCOMPLETE (la résolution d'adresse est en progression et l'adresse MAC n'est pas encore déterminée)
    b. REACHABLE (le voisin a été rejoint récemment)
    c. STALE (le voisin n'est pas connu comme étant accessible mais jusqu'à ce que du trafic lui soit envoyé, aucune tentative ne va être faite pour vérifier son accessibilité)
    d. DELAY (le voisin n'est plus connu comme étant accessible, mais du trafic lui a été récemment envoyé. On retarde un certain temps avant de vérifier l'accessibilité)
    e. PROBE (le voisin n'est plus connu comme étant accessible, et des messages de sollicitation lui ont été envoyées pour le vérifier)
6. compteur d'envoi de sollicitation. Il est incrémenté à chaque envoi et ce, jusqu'à MAX_MULTICAST_SOLICIT. Cela sert à déterminer si l'on arrive à rejoindre un nœud.
7. Un timer. La première fois qu'un noeud envoie des paquets à un voisin dont l'état de l'entrée est à STALE, il doit changer cet état pour DELAY. Si après DELAY_FIRST_PROBE_TIME secondes, cette entrée est toujours à DELAY, elle passe à PROBE. Si une confirmation est reçue, elle passe à REACHABLE. Ce timer sert à déterminer l'état de l'entrée. Ensuite, quand on entre à l'état PROBE, le nœud envoie des sollicitations à toutes les retransTimer (quand le Timer arrive à zéro), jusqu'à obtention de la confirmation. Si après MAX_UNICAST_SOLICIT sollicitations envoyées, aucune confirmation n'est reçue, on arrête et on efface l'entrée. Une fois que l'entrée est à PROBE, le timer compte le temps de retransmission.

8. compteur. Ce compteur sert à compter le nombre d'envois de sollicitations. Il est donc incrémenté jusqu'à MAX_UNICAST_SOLICIT. Si aucune confirmation n'est reçue, on arrête et on efface l'entrée.
9. un identificateur de l'interface utilisé.

3.2.2.2 Liste de routeurs par défaut

Cette liste est une liste de routeurs où les paquets peuvent être envoyés. Elle est composée de :

1. un pointeur sur les entrées de la « Neighbor Cache » ;
2. une durée de vie associée à ce routeur ;
3. timer : intervalle auquel un AR envoie des RA non sollicités ;
4. timer : indique le temps en millisecondes où le nœud peut assurer que le routeur est rejoignable ;
5. drapeau indiquant si ce routeur travaille comme un HA sur le lien.

3.2.2.3 Liste des associations d'adresse

Cette liste fait l'association entre l'adresse CoA et l'adresse nominale du MN. Elle est maintenue au niveau des HA et des MAP (dans ce cas, elle fait l'association RCoA ⇔ LCoA). Les entrées de cette liste sont identifiées par l'adresse nominale. Dans le cas d'associations simultanées, on a deux entrées dans cette liste. On reconnaît une association simultanée quand deux entrées ont la même adresse nominale mais des adresses CoA différentes. Cette liste est composée des champs suivants :

1. l'adresse nominale (ou RCoA du MN) du MN,
2. le CoA (ou LCoA) du MN,
3. l'adresse du HA (ou du MAP) servant le MN,
4. la durée de vie restante de cette association,
5. un drapeau indiquant si cette entrée est un enregistrement primaire au HA,
6. la valeur maximale du champ numéro de séquence (« Sequence Number ») reçu dans les BU précédents,

7. l'information d'utilisation pour cette entrée,
8. l'identifiant de l'interface associée à cette entrée,
9. l'identificateur du tunnel associé à cette entrée,
10. un identificateur de la table des informations de transmission (FIB).

Une entrée non marquée comme étant un enregistrement primaire peut être remplacée à n'importe quel moment suivant la politique locale de remplacement mais ne devrait pas être supprimée inutilement.

3.2.2.4 Liste de HA ou de MAP

Chaque HA (MAP) doit maintenir une liste des HA (MAP). Cette liste contient les informations à propos de tous les routeurs sur le même lien que le sien qui agissent comme HA (MAP). Chaque HA (MAP) maintient une liste séparée pour chaque lien où il agit comme HA (MAP). Un routeur est connu pour agir comme HA s'il envoie des *Router Advertisement* (RA) avec le bit H à 1. Cette liste ressemble à la liste de routeurs. Une nouvelle entrée est créée ou mise à jour sur réception des RAs. La liste contient les champs suivants :

1. l'adresse IP du HA (MAP) (adresse source du « *Router Advertisement* »),
2. le préfixe (pour création d'adresse),
3. une ou plusieurs adresses globales de ce HA (MAP) (apprises par l'option d'information de préfixe),
4. la durée de vie restante de cette entrée,
5. la préférence de ce HA (plus la valeur est grande, plus la préférence est élevée),
6. le mode d'opération (drapeau RIP et V de l'option MAP).

3.2.2.5 Liste de gestion des tunnels

Le *tunneling* est une technique consistant à établir un lien virtuel entre deux nœuds pour transmettre des paquets. Du point de vue des nœuds, ce lien apparaît comme étant point à point. Le premier nœud encapsule le paquet orignal et le

transmet sur ce tunnel. Le second nœud dé-encapsule le paquet reçu. La liste pour la gestion du tunnel doit contenir les champs suivants :

1. Adresse du nœud d'entrée du tunnel,
2. Adresse du nœud de sortie du tunnel,
3. durée de vie du tunnel,
4. identificateur de l'interface et
5. durée de vie.

3.2.2.6 Liste de préfixe agrégé

Quand le MN est loin de son réseau d'origine, il reçoit des messages *Prefix Advertisement* lui donnant de l'information sur son lien d'origine. Les routeurs surveillent ces messages et construit une liste agrégée et la donne au MN. De plus, un HA (MAP) est obligé d'envoyer de nouvelles informations sur ces préfixes quand la durée de vie valide ou préférée ou l'état des bits a changé pour le préfixe pour lequel le MN a enregistré son adresse nominale. Si le MN envoie une sollicitation, il faut lui retourner cette liste. Les routeurs envoient des RA qui indiquent si l'expéditeur est disposé à être un routeur par défaut. Ces messages contiennent aussi les options « *Prefix Information* » qui donnent une liste de l'ensemble des préfixes qui identifient l'adresse IP du lien. Une structure pour la liste de préfixe agrégé est donc définie. Chaque entrée contient les champs suivants :

1. préfixes,
2. durée de vie valide (en seconde) pour laquelle le préfixe demeure valide (0xffffffff = infini),
3. temps de vie préféré (en seconde) pour l'adresse générée à partir de ce préfixe.

Chacune des entrées est contenue dans une structure d'entrée. Cette structure contient les champs suivants :

1. Nombre d'entrées,
2. Tableau de liste de préfixe agrégé (de taille nombre d'entrée),

3. Timer : Si aucun *Prefix Advertisement* n'a été envoyé dans les dernières MAXMOBPFXADVINTERVAL, il faut s'assurer qu'une transmission est prévue,
4. Timer : délai avant lequel la liste va être envoyé. Le HA devrait continuer à envoyer les *Advertisements* non sollicités au MN jusqu'à ce qu'il y ait acquittement,
5. Temps avant la première retransmission s'il n'y a pas eu acquittement. Ce temps est doublé à chaque retransmission, jusqu'à ce qu'on ait fait PREFIXE_ADV_RETRIES retransmission ou que le l'association expire.
6. Nombre de retransmission. Quand c'est égal à PREFIXE_ADV_RETRIES, on arrête.

3.2.2.7 Liste de CoA (« CoA pool »)

Cette liste est optionnelle. À chaque déplacement, le MN doit générer un nouveau CoA. Pour tester l'unicité d'une adresse, on doit faire un DAD avant de pouvoir assigner cette adresse à l'interface à laquelle on s'attache. Si le DAD s'effectue avec succès, on enregistre le CoA au HA et aux CN. Si le DAD échoue, le MN doit générer une nouvelle adresse et on doit refaire une détection de duplication d'adresse. Le DAD peut consommer beaucoup de temps particulièrement lorsque le MN roule des applications en temps réel. Pendant la détection de duplication d'adresse, toutes les communications du MN sont interrompues.

On propose une méthode pour minimiser les coupures en découplant le DAD de la relève de niveau 3. Cela est effectué en configurant un nouveau CoA en avance qui pourra être utilisé sans égard au DAD après que le MN se soit déplacé. Les AR génèrent aléatoirement plusieurs CoA en avance et testent l'unicité de ces adresses à travers la procédure classique de DAD. Ensuite, on peut mettre ces adresses dans un *pool CoA*. Pour chaque adresse dans le pool, le AR agit comme un proxy passif pour ne pas affecter la *destination cache* et le *neighbor cache*. Le proxy doit créer un CoA, faire un DAD et conserver l'adresse pour les différents MN. Si le proxy se rend compte qu'un nœud est en train de configurer une adresse qui est contenue dans le pool CoA, il doit abandonner cette adresse et en configurer une autre.

Les CoA configurés en avance sont conservés au niveau de chaque AR. Le MN peut ainsi acquérir un nouveau CoA rapidement en le demandant au AR sans qu'aucun DAD ne soit effectué.

Le AR doit maintenir un *pool CoA* pour chaque interface (préfixe différent). Le nombre d'adresses générées initialement est au moins CAPACITY_OF_POOL par *pool CoA*. Par défaut, cette capacité est de 100, mais elle peut être configurable. Un DAD doit être exécuté sur l'adresse générée. Si l'adresse est déjà utilisée, elle ne doit pas être mise dans le pool.

Dès qu'une adresse DAD-free NCoA (adresse CoA pour laquelle un DAD a été effectué avec succès) est conservé dans le pool, le AR devient un proxy passif pour cette adresse. Il ne doit pas envoyer de *Neighbor Advertisement* non sollicité avec cette adresse. Il ne doit pas non plus répondre à un *Neighbor Sollicitation* avec le champ adresse cible mis à la valeur d'une adresse contenue dans le *pool CoA*. Quand le AR reçoit un *Neighbor Sollicitation* demandant l'exécution d'un DAD, il doit vérifier si l'adresse cible contenue dans le message correspond à une des adresses dans le *pool CoA*. Si cette adresse est contenue dans le pool, il doit l'en enlever.

Dès que le MN se connecte sur un nouveau lien, il peut envoyer un *Router Sollicitation* à une adresse multicast pour tous les routeurs. S'il veut un DAD-free NCoA, il doit inclure l'option DAD-free NCoA Request.

Une liste maintenue au niveau des AR doit donc être définie. Elle contient le DAD-free CoA. Un tableau de liste est lui-même contenu dans une structure qui contient les champs suivants :

1. Préfixe de l'interface,
2. Nombre de DAD-free CoA,
3. Tableau de DAD-free CoA,
4. Capacité du pool (100 par défaut),
5. Identificateur d'interface.

3.3 Achèvement des callbacks

Comme on l'a vu au chapitre 2, les API sont asynchrones. Les applications utilisant les registres de *callback* sont appelées après qu'on ait complété un appel. Une application peut, par exemple, décider d'appeler une fonction de mise à jour d'association d'adresse. Cette mise à jour ne se fera pas instantanément. L'application appelle la fonction mais continue son travail initial. Quand la mise à jour d'association d'adresse est terminée, l'application recevra un *callback* l'avertissant du fait que la mise à jour s'est déroulée avec succès.

3.3.1 Type de callback

Chaque *callback* est identifié par une valeur spécifiant le type de *callback*. On définit un type différent pour chacune des opérations suivantes :

- création d'une table ou d'un identificateur pour une table de destinataire,
- effacement d'un identificateur pour une table de destinataire,
- ajout d'une ou de plusieurs entrées dans une table de destinataire,
- effacement d'une ou de plusieurs entrées dans une table de destinataire,
- effacement d'une table de destinataire,
- demande d'attribut pour une table de destinataire,
- création d'une table ou d'un identificateur de table de gestion de tunnel,
- effacement d'un identificateur pour une table de gestion de tunnel,
- ajout d'une ou de plusieurs entrées dans une table de gestion de tunnel,
- effacement d'une ou de plusieurs entrées dans une table de gestion de tunnel,
- effacement d'une table de gestion de tunnel,
- demande d'attribut pour une table de gestion de tunnel,
- création d'une table ou d'un identificateur de table d'association d'adresse,
- effacement d'un identificateur pour une table d'association d'adresse,
- ajout d'une ou de plusieurs entrées dans une table d'association d'adresse,
- effacement d'une ou de plusieurs entrées dans une table d'association d'adresse,
- effacement d'une table d'association d'adresse,

- demande d'attribut pour une table d'association d'adresse.

Cette énumération détaille les types de callback que l'API peut générer pour signaler l'achèvement d'une fonction asynchrone particulière.

3.3.2 Réponses asynchrones

Une réponse asynchrone contient

- un identificateur d'interface,
- un type de *callback*,
- un code d'erreur pour cette réponse et
- une structure de données spécifique à la fonction appelée.

La structure contenant la réponse asynchrone est elle-même contenue dans une autre structure pouvant contenir plusieurs réponses. Ceci est utile pour une fonction qui demande plusieurs actions simultanément. Cette structure mère contient :

- le nombre de réponses contenues dans le tableau de réponse (n_resp),
- le tableau de réponse, et
- un drapeau indiquant si tout s'est déroulé avec succès (allOk).

Le drapeau ainsi que le nombre de réponses ont des significations différentes suivant ce qu'ils indiquent :

1) La fonction appelée n'a réalisé qu'une seule opération :
   a) n_resp = 1, le tableau de réponse ne contient qu'un seul élément.
   b) allOK = TRUE, si l'opération demandée s'est déroulée avec succès et que la seule réponse de retour est le code de réponse et non une structure spécifique à l'opération demandée.
   c) if allOK = FALSE, la structure de réponse contient le code d'erreur.
2) La fonction a été appelée pour réaliser plusieurs opérations :
   a) Tout s'est déroulé avec succès en même temps et la seule valeur retournée est le code de réponse :
      i) allOK = TRUE, n_resp = 0 (pas de structure spécifique).
   b) Tout s'est déroulé avec succès en même temps et la valeur de retour est contenue dans le tableau de réponse :

i) allOK = TRUE, n_resp = nombre de réponses dans le tableau.

c) Certaines opérations sont complétées mais pas toutes :
   i) allOK = FALSE, n_resp = le nombre de réponses complétées.
   ii) Le tableau de réponse contient un élément pour chaque réponse complétée. Celui-ci contient un code d'erreur indiquant si l'opération s'est déroulée avec succès.

D'autres types d'actions asynchrones peuvent se produirent. Ces actions correspondent à des notifications d'évènements.

3.3.3 Notifications d'évènements

Les notifications d'évènements surviennent suite à des évènements qui n'ont pas été appelés par une fonction. On peut retrouver par exemple l'absence d'une table ou l'arrivée d'un paquet particulier sur une interface. Les structures définies dans les sections suivantes servent à définir les notifications d'évènements.

3.3.3.1 Types de notifications d'évènement

On doit donner un code unique à chaque type d'évènement qui peut se produire de manière à ce que l'application puisse reconnaître ces types d'évènement. Les évènements que l'on retrouve sont les suivants :

- Absence d'une table de destinataire,
- Absence d'une table de gestion de tunnel,
- Absence d'une table d'association d'adresse,
- Une entrée dans une table de gestion de tunnel est sur le point d'expirer,
- Une entrée dans une table de destinataire est sur le point d'expirer,
- Une entrée dans une table d'association d'adresse est sur le point d'expirer,
- Une entrée dans une table de gestion de tunnel est incomplète,
- Une entrée dans une table de destinataire est incomplète,
- Une entrée dans une table d'association d'adresse est incomplète,
- Arrivée d'un *Binding Update,*

- Arrivée d'un *Binding Refresh Request,*
- Arrivée d'un *Fast Binding Update,*
- Arrivée d'un *Local Binding Update,*
- Arrivée d'un *Handover Initiate,*
- Arrivée d'un *Handover to Third.*

Ce type est compris à l'intérieur d'une structure permettant de définir les caractéristiques de l'évènement.

3.3.3.2 Données de l'évènement

Cette structure contient le type d'évènement ainsi que la structure spécifique à un évènement particulier. Cette structure est retournée par un *Callback* de notification d'évènement sous forme d'un tableau. Cela permet d'envoyer plusieurs évènements en même temps. La structure des données de l'évènement doit contenir les informations suivantes :

- Le type d'évènement,
- L'identificateur de l'interface où l'évènement s'est produit,
- Le contexte d'utilisation, et
- Une union contenant les structures de données spécifiques aux évènements possibles.

Comme on l'a déjà spécifié, cette structure est contenue à l'intérieur d'une autre structure mère comprenant le nombre d'évènements ainsi qu'un tableau de données d'évènements. Le tableau doit contenir le nombre d'évènements spécifié.

3.4 Fonctions reliées à la gestion de la relève

L'application enregistre les fonctions manipulant les réponses asynchrones à l'exécution de l'API. La fonction d'achèvement de service (*Callback*) est implémentée par l'application et est enregistrée à l'exécution de l'API par la fonction de NPF_Register(). L'application peut recevoir exactement le même nombre de

réponses que la quantité demandée, mais les réponses peuvent être des invocations multiples de fonction. La structure de données d'achèvement des *callback* contient un tableau de réponses, de sorte que des appels de fonction pour des opérations multiples puissent être agrégés dans peu d'invocations de fonction (peut-être juste une). Les choix concernant l'implémentation de l'API et l'assignation des réponses aux invocations des fonctions sont laissés aux programmeurs.

La première fonction à utiliser est une fonction d'enregistrement. Cette fonction est utilisée par une application pour enregistrer sa fonction d'achèvement de *callback* permettant de recevoir des réponses asynchrones liées aux appels de fonction de l'API. L'application peut enregistrer de multiples fonctions d'achèvement de *callback* en utilisant cette fonction. La fonction d'achèvement de *callback* est identifiée par la paire *userContext* (contexte d'utilisation) et *callbackFunc* (la fonction), et pour chaque paire individuelle, un *callbackHandle* (identificateur de callback) unique sera assigné pour référence future. L'identificateur (*callbackHandle*) sera utilisé par l'application pour indiquer quel *callback* appeler. Il sert aussi pour dé-enregistrer une paire *userContext* et de *callbackFunc*. Puisque la fonction est identifiée par le *userContext* et le *callbackFunc*, on permet l'enregistrement double de la même fonction avec un *userContext* différent. En outre, le même *userContext* peut être commun pour différentes fonctions. L'enregistrement double de la même paire *userContext* et *callbackFunc* n'a aucun effet et produira un identificateur qui est déjà assigné à la paire. Cette fonction d'enregistrement est une fonction synchrone et n'a aucun rappel d'accomplissement relié.

Évidemment, une fonction de dé-enregistrement existe aussi. Cette fonction est utilisée par une application pour dé-enregistrer une fonction de *callback*. Cette fonction est également synchrone. Le *callbackHandle* est l'identificateur unique représentant la paire *userContext* et *callbackFunc* à dé-enregistrer. Ensuite, aucun appel de fonction utilisant cet identificateur ne peut être fait.

Pour chaque table à manipuler, on a des fonctions servant à :

- créer une table ou un identificateur de table,
- effacer un identificateur pour une table,
- ajouter une ou plusieurs entrées dans une table,

- effacer une ou plusieurs entrées dans une table,
- effacer toute une table,
- demander les attributs pour une table.

Par exemple, une fonction est utilisée pour ajouter une ou plusieurs entrées à une table. Si aucune entrée n'existe pour l'adresse spécifiée dans la structure concernée, alors une nouvelle entrée est créée, sinon il s'agit d'une mise à jour. Cette fonction possède les paramètres suivants :

- Le *callbackHandle* : sert à identifier le *callback*.
- Le *correlator* : identifie le contexte d'application pour cet appel. Il permet de distinguer une invocation de fonction lors d'appels multiples d'une même fonction.
- L'erreur : indique si l'application souhaite recevoir un rappel de service d'accomplissement. Une application pourrait décider de ne pas recevoir de confirmation sur le fait que l'action a été réalisée.
- Un *tblHandle* : identificateur de la table affectée.
- Un *numEntries* : le nombre d'entrées à ajouter à la table.
- Un *entryArray* : Un tableau des entrée à ajouter à la table.

La plupart des appels de fonction peuvent être construits sur ce modèle. Une fois que le *callbackHandle* est créé, il suffit de donner un identificateur de la table et un tableau d'entrée à ajouter pour mettre à jour ou ajouter des entrées.

Le Tableau 3.1 montre les différentes fonctions pour la gestion des relèves ainsi que les structures de *callback* retournées.

Tableau 3.1 Fonctions de gestion des relèves et structures retournées

Nom de la fonction	Type	Union retournée
NPF_MIPV6DestTableHandleCreate	NPF_MIPV6_DEST_TABLE_HANDLE_CREATE	NPF_MIPV6_Dest_Tbl_Handle_Create_t
NPF_MIPV6DestTableHandleDelete	NPF_MIPV6_DEST_TABLE_HANDLE_DELETE	NPF_MIPV6_Dest_Tbl_Handle_Delete_t
NPF_MIPV6DestEntryAdd	NPF_MIPV6_DEST_ENTRY_ADD	NPF_MIPV6DestEntryResp_t
NPF_MIPV6DestEntryDelete	NPF_MIPV6_DEST_ENTRY_DELETE	NPF_MIPV6DestEntryResp_t
NPF_MIPV6DestTableFlush	NPF_MIPV6_DEST_TABLE_FLUSH	Aucune
NPF_MIPv6TunnelTableHandleCreate	NPF_MIPv6_TUNNEL_TABLE_HANDLE_CREATE	NPF_MIPv6_Tunnel_Tbl_Handle_Create_t
NPF_MIPv6TunnelTableHandleDelete	NPF_MIPv6_TUNNEL_TABLE_HANDLE_DELETE	NPF_MIPv6_Tunnel_Tbl_Handle_Delete_t
NPF_MIPv6TunnelEntryAdd	NPF_MIPv6_TUNNEL_ENTRY_ADD	NPF_MIPV6_TunnelEntryResp_t
NPF_MIPv6TunnelEntryDelete	NPF_MIPv6_TUNNEL_ENTRY_DELETE	NPF_MIPV6_TunnelEntryResp_t
NPF_MIPv6TunnelTableFlush	NPF_MIPv6_TUNNEL_TABLE_FLUSH	Aucune
NPF_MIPv6BindingTableHandleCreate	NPF_MIPv6_BINDING_TABLE_HANDLE_CREATE	NPF_MIPv6_Binding_Tbl_Handle_Create_t
NPF_MIPv6BindingTableHandleDelete	NPF_MIPv6_BINDING_TABLE_HANDLE_DELETE	NPF_MIPv6_Binding_Tbl_Handle_Delete_t
NPF_MIPv6BindingEntryAdd	NPF_MIPv6_BINDING_ENTRY_ADD	NPF_MIPV6_BindingEntryResp_t
NPF_MIPv6BindingEntryDelet	NPF_MIPv6_BINDING_ENTRY_DELETE	NPF_MIPV6_BindingEntryResp_t
NPF_MIPv6BindingTableFlush	NPF_MIPv6_BINDING_TABLE_FLUSH	Aucune

3.5 Fonctions reliées à la gestion de l'interface

Ces fonctions utilisent le même principe que celui décrit à la section précédente. On y retrouve une fonction d'enregistrement et un *callbackHandle* servant à identifier un *callback* particulier. L'API de gestion des interfaces [22] comprend les fonctions suivantes :

- NPF_IfMIPv6HAEnable : cette fonction permet à une interface MIPv6 de pouvoir agir comme un HA. Une adresse IPv6 doit avoir été assignée préalablement.

- NPF_IfMIPv6HADisable : cette fonction dé-active une fonctionnalité HA à une interface MIPv6.
- NPF_IfMIPv6AREnable : cette fonction permet à une interface MIPv6 de pouvoir agir comme un AR. Une adresse IPv6 doit avoir été assignée préalablement.
- NPF_IfMIPv6ARDisable : cette fonction dé-active une fonctionnalité AR à une interface MIPv6.
- NPF_IfMIPv6MAPEnable : cette fonction permet à une interface MIPv6 de pouvoir agir comme un MAP. Une adresse IPv6 doit avoir été assignée préalablement.
- NPF_IfMIPv6MAPDisable : cette fonction dé-active une fonctionnalité MAP à une interface MIPv6.
- NPF_IfMIPv6AddrSet : cette fonction assigne une ou plusieurs adresses IPv6 à une interface MIPv6. Si l'interface a déjà une ou plusieurs adresses, elles sont enlevées et remplacées par le nouvel ensemble. Si le nombre d'adresses est nul, toutes les adresses sont enlevées.
- NPF_IfMIPv6AddrAdd : cette fonction ajoute des adresses MIPv6 à une interface. Si l'interface a déjà une ou plusieurs adresses, celles-ci ne sont pas enlevées.
- NPF_IfMIPv6AddrDelete : cette fonction supprime les adresses d'une interface MIPv6.
- NPF_IfMIPv6FibSet : cette fonction associe un tableau d'expédition IPv6 (FIB) à une ou plusieurs interfaces MIPv6. Le nouvel identificateur de FIB devient un attribut de l'interface. Un seul FIB peut être associé à plusieurs interfaces.

Ces fonctions suivent une convention permettant à plusieurs identificateurs d'interface d'être passés pour l'action dans une seule invocation de fonction. Dans chaque cas, il y a un argument indiquant la taille du tableau d'identificateur d'interface. Aucune limite sur la taille des tableaux n'est indiquée. Toutefois, une implémentation peut imposer la limite de son choix.

Le Tableau 3.2 montre les différentes fonctions pour la gestion des interfaces ainsi que les structures de *callback* retournées.

Tableau 3.2 Fonctions d'interface et structures retournées

Nom de la fonction	Type de callback	Structure retournée
NPF_IfMIPv6HAEnable	NPF_IF_MIPV6_HA_ENABLE	NPF_MIPv6_HAEnable_t
NPF_IfMIPv6HADisable	NPF_IF_MIPV6_HA_DISABLE	NPF_MIPv6_HAEnable_t
NPF_IfMIPv6AREnable	NPF_IF_MIPV6_AR_ENABLE	NPF_MIPv6_AREnable_t
NPF_IfMIPv6ARDisable	NPF_IF_MIPV6_AR_DISABLE	NPF_MIPv6_AREnable_t
NPF_IfMIPv6MAPEnable	NPF_IF_MIPV6_MAP_ENABLE	NPF_MIPv6_MAPEnable_t
NPF_IfMIPv6MAPDisable	NPF_IF_MIPV6_MAP_DISABLE	NPF_MIPv6_MAPEnable_t
NPF_IfMIPv6AddrSet	NPF_IF_MIPV6_ADDR_SET	NPF_MIPv6Add_t
NPF_IfMIPv6AddrAdd	NPF_IF_MIPV6_ADDR_ADD	NPF_MIPv6Add_t
NPF_IfMIPv6AddrDelete	NPF_IF_MIPV6_ADDR_DELETE	Aucune
NPF_IfMIPv6FibSet	NPF_IF_MIPV6_FIB_SET	Aucune

CHAPITRE IV

IMPLÉMENTATION ET RÉSULTATS

Dans ce chapitre, nous allons proposer une implémentation de l'API présentée aux chapitres précédents. Cette implémentation a pour objectif de faire une validation fonctionnelle sur un système réel. Pour cela, nous utilisons un HA existant qui sera transformé pour être séparé en deux blocs : un bloc FE et un bloc CE pour y insérer l'API. Dans un premier temps, nous analyserons la structure du HA, ce qui nous permettra de définir dans quelles conditions se réalisera l'implémentation. Une fois l'environnement de l'implémentation présenté, nous nous focaliserons sur la mise en œuvre de l'architecture. Finalement, nous conclurons par une évaluation de notre proposition.

4.1 Présentation du système

Plusieurs implémentations du protocole MIPv6sont disponibles :

- MIPL (Mobile IPv6 for Linux (http://www.mobile-ipv6.org))
- KAME (http://www.kame.net/)
- Microsoft Mobile IPv6 Implémentation (http://www.research.microsoft.com/programs/europe/projects/MIPv6.asp)
- LANCASTER MOBILE IPv6 PACKAGE (http://www.cs-ipv6.lancs.ac.uk/ipv6/MobileIP).

Parmi ces différentes implémentations, nous avons choisie MIPL, pour plusieurs raisons : elle est disponible gratuitement, de nouvelles versions sortent régulièrement permettant de suivre l'évolution des protocoles, elle est utilisée par un grand nombre de personnes. Par conséquent, elle dispose d'un support. Le code MIPL est uniquement basé sur le protocole MIPv6 et bien que l'API couvre la suite des protocoles MIPv6 (c'est-à-dire HMIPv6, FMIPv6 et le bicasting), l'implémentation du prototype sera partielle mais permettra tout de même de vérifier l'essentiel de l'API. Pour voir comment l'implémentation sera faite, nous devons analyser le fonctionnement du code MIPL.

4.1.1 Détails d'environnement

Comme nous l'avons déjà vu, l'implémentation de l'API est basée sur l'architecture décrite à la Figure 4.1. Un système de base se doit d'avoir les caractéristiques suivantes :

- Un RTM (Routing Table Manager) qui gère le RIB (Routing Information Base) au niveau contrôle. Il interagit avec les protocoles de routage pour mettre à jour le RIB.
- Le RTM utilise les fonctions définies dans l'API pour mettre à jour le FIB (Forwarding Information Base) au niveau du FE.
- La structure et l'organisation du RIB ne sont soumises à aucune règle. Chacun peut l'implémenter à sa manière. La structure du RIB n'est pas connue de l'API.

Quand un paquet arrive sur une interface, il est examiné par le niveau transport pour déterminer s'il est destiné ou non à cet hôte. Si c'est le cas et s'il s'agit d'un paquet de contrôle, il est transmis au CE. Là, le RTM gère l'information contenue dans ce paquet pour la mise à jour des différentes tables. Le RTM met à jour le RIB et appelle les fonctions de l'API pour la mise à jour du FIB. Le FE peut ensuite retourner un signal au CE indiquant si l'opération s'est déroulée avec succès ou non.

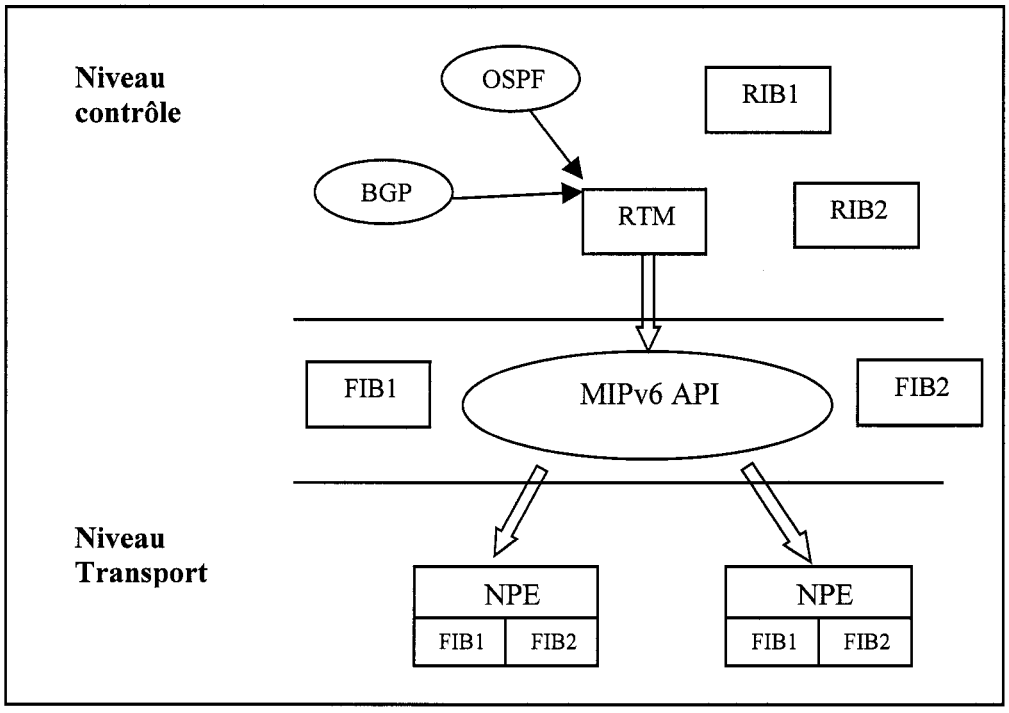


Figure 4.1 Architecture de l'implémentation

Le code MIPL-1.0 consiste en un patch sur le noyau 2-4.25. Il s'insère à l'intérieur de la pile IP du noyau Linux. De manière à construire un CE et un FE fonctionnel, nous devons analyser le cheminement des paquets au niveau du noyau et ainsi voir comment MIPv6 s'insère dans le noyau 2-4.

4.1.2 Cheminement des paquets dans la pile IPv6

Un paquet IPv6 arrive à un hôte par la carte réseau qui correspond à la couche physique du modèle OSI. Lorsque la carte réseau reçoit un paquet, une interruption est déclenchée et un tampon socket est alloué. Après avoir mis le tampon dans la bonne queue, celui-ci en est enlevé et est traité par le paquet « *handler* » approprié (fonction correspondant au protocole du paquet entrant). Si le paquet est un paquet IPv6, on parle du « IPv6 packet handler ».

La fonction « IPv6 packet handler » est *ipv6_rcv()*. Après quelques vérifications (checksum IP...), les « hooks Netfilter » sont appelés. Netfilter fournit

une interface abstraite et générique au code de routage standard. Cette interface est utilisée pour du filtrage de paquet, du découpage, du NAT et pour mettre en queue les paquets destinés en « userspace ». La Figure 4.2 indique les différents « hook » (point d'ancrage de Netfilter).

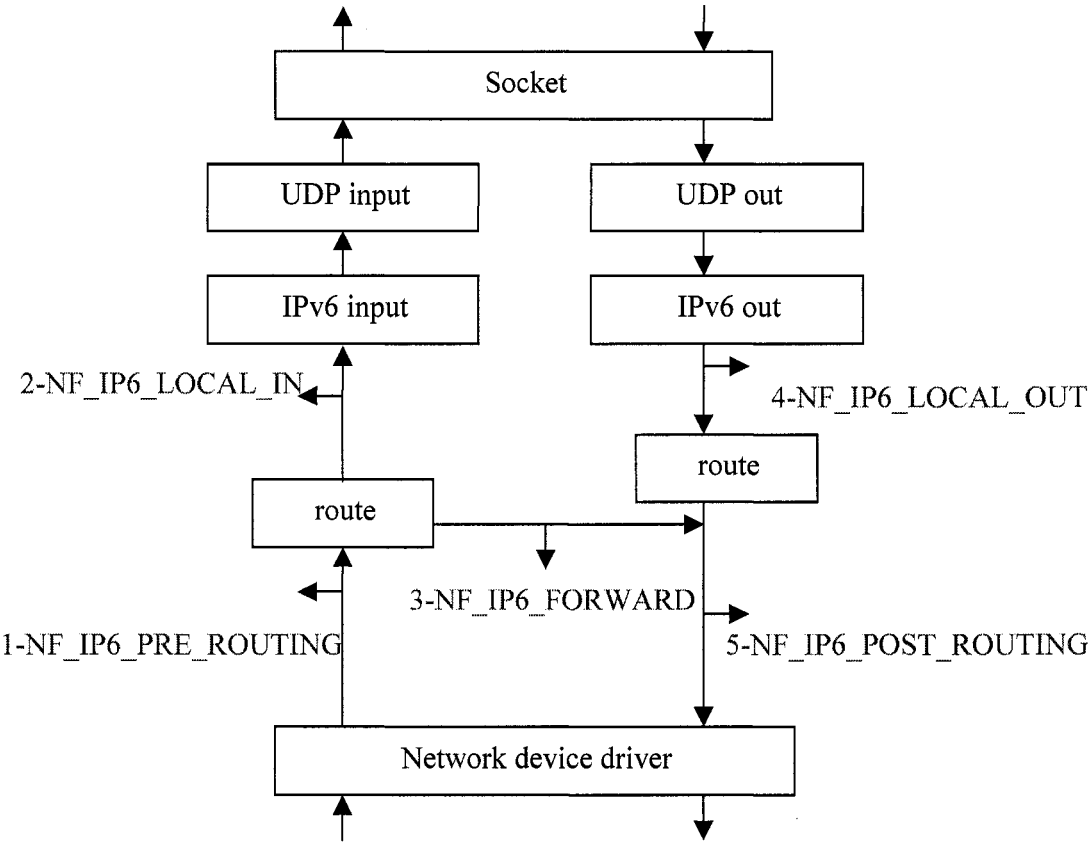


Figure 4.2 Les Hooks Netfilters

Netfilter sert à modifier les paquets dans le noyau. La pile réseau fournit des points d'ancrage sur le chemin des paquets. Les fonctions du noyau voulant modifier les paquets s'enregistrent pour être appelés à certains points. À chacun des cinq points définis, le protocole va appeler Netfilter et lui passer le paquet. Sur chacun de ces points, on peut accrocher n'importe quelle fonction. Quand Netfilter reçoit un paquet, il va appeler chacune des fonctions. Les fonctions vont pouvoir examiner le paquet, le modifier, s'en débarrasser, l'autoriser à passer ou le mettre en queue pour le « user-space ».

Si le paquet passe le premier point d'ancrage (NF_IP6_PRE_ROUTING), il est envoyé vers la fonction *ipv6_rcv_finish()*. Là on vérifie la destination finale du paquet. On fait le tour dans la table de routage du noyau (FIB) pour savoir quoi faire avec ce paquet. Suivant l'information recueillie, le paquet est traité par une des quatre fonctions d'entrée possible :

1. *ip6_pkt_discard()* : jette paquet ;
2. *ip6_mc_input()* : paquet multicast entrant ;
3. *ip6_forward()* : destination finale n'est pas cet hôte on doit l'envoyer vers son prochain saut ;
4. *ip6_input()* : livre le paquet pour cet hôte; nous devons traiter le prochain en-tête du paquet.

Si le paquet est destiné à cet hôte, on arrive au point d'ancrage deux (NF_IP6_LOCAL_IN) comme indiqué à la Figure 4.2. Là, le paquet est démultiplexé et envoyé à tour de rôle à plusieurs fonctions, suivant les différentes en-têtes contenus dans le paquet. Parmi ces fonctions, on retrouve entre autres :

1. *mip6_mh_rcv()* : qui s'occupe des en-têtes de mobilité MIPv6;
2. *icmpv6_rcv()* : pour les messages icmp (comme les messages de découverte d'adresse des HA...);
3. *ipv6_destopt_rcv()* : pour les options de destination;
4. *ipv6_rthdr_rcv()* : pour les messages d'en-tête de routage;
5. *ip6ip6_rcv()* : pour la gestion des tunnels;
6. *xfm6_rcv()* : pour la gestion de IPSec (AH et ESP);
7. *tcp_v6_rcv()*;
8. *udpv6_rcv()* ...

La Figure 4.3 illustre le cheminement du paquet dans le noyau au niveau de la couche trois du modèle OSI.

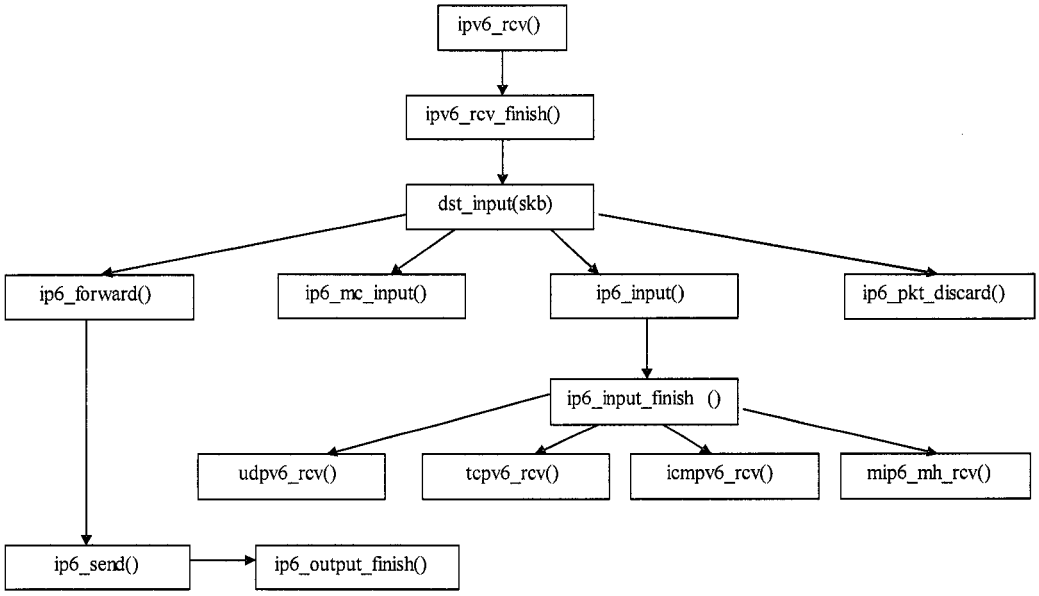


Figure 4.3 Cheminement du paquet à la couche 3 du noyau

Le code MIPL-1.0 s'insère à l'intérieur de la pile IPv6 de manière à traiter les paquets de mobilité. Dans la section suivante, nous allons analyser plus en détail son fonctionnement.

4.1.3 Fonctionnement MIPL

Le code MIPL peut servir pour différentes sortes de nœuds. Un nœud peut être soit HA, MN ou CN. Puisque l'API est développé pour un routeur, nous allons uniquement regarder ici le fonctionnement du HA.

Le HA accepte les enregistrements de CoA primaire venant du nœud mobile. Si le MN est à l'extérieur de son réseau d'origine, le HA intercepte les paquets destinés au MN et les transmet à l'adresse CoA du MN via un tunnel. Le système complet est implémenté en un seul module du noyau. Le codage au niveau noyau rend plus difficile le déboguage mais permet plus de flexibilité. Puisque l'on peut insérer ou enlever dynamiquement le module MIPv6, on ne peut pas utiliser des appels directs du noyau (IPv6) vers les fonctionnalités du module MIPv6. Ainsi, un module (MIPGlue) a été ajouté pour permettre l'interaction entre les modules IPv6 et MIPv6. La Figure 4.4 illustre l'architecture générale du noyau.

Le standard MIPv6 spécifie de nouvelles options de destinations que tous les nœuds doivent être en mesure de comprendre :

1. le « Binding Update »,
2. le « Binding Acknowledgement »,
3. le « Binding Request » et
4. le « Home Address destination option ».

Des pointeurs pour les fonctions traitant ces options doivent être ajoutés à l'intérieur de l'implémentation du module IPv6 et doivent être spécifiés dans le module « MIPGlue ».

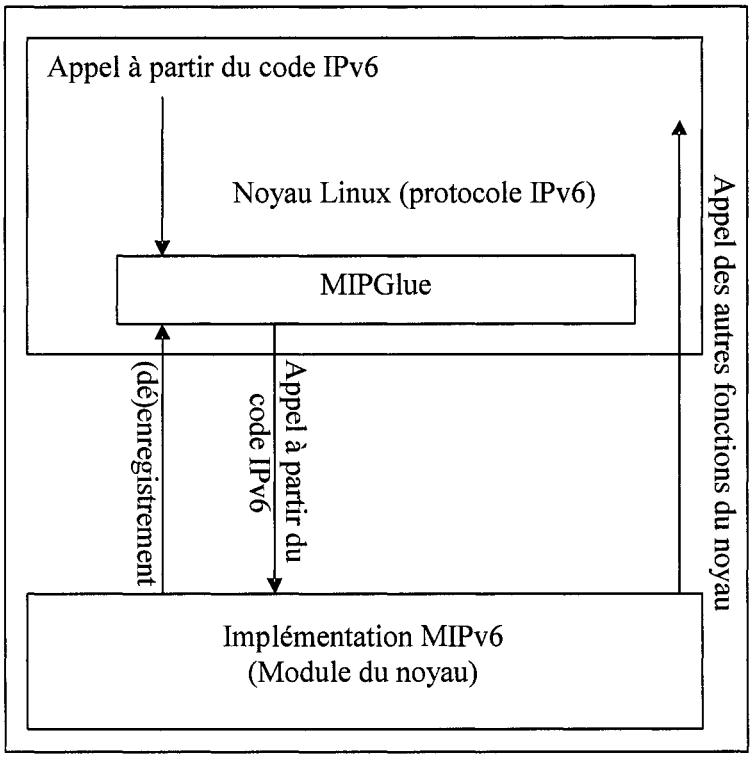


Figure 4.4 Architecture générale

Les structures de données principales à MIPL sont :

1. La « Binding Cache ». Cette structure est maintenue au niveau de tous le nœuds pour sauvegarder les associations d'adresse valides ;

2. La « Binding Update Liste ». Cette structure est maintenue uniquement par les MN. Elle contient l'information concernant chaque message d'association d'adresse transmis par ce nœud ;
3. La « Home Agent List ». Cette liste est maintenue par les HA et sert à conserver l'information à propos des autres HA sur le lien où ce nœud sert de HA. Cette liste est utilisée pour le mécanisme de découverte automatique de HA.

Quand un nœud sert comme HA, il doit réagir aux messages d'association d'adresse qu'il reçoit. Le HA vérifie alors la validité du message et s'il est capable de traiter la demande, (ressources suffisantes) il commence alors à agir comme HA pour le MN. Il crée le tunnel, enregistre la route dans la table de routage du noyau, met à jour la « Binding Cache » et transmet un acquittement au nœud mobile à l'origine du message. Le HA va alors intercepter les paquets destinés au MN et les transmettre par le tunnel.

La Figure 4.5 illustre le cas où un HA reçoit un message d'association d'adresse et répond avec un acquittement. Les pointeurs pour les fonctions de traitement des paquets MIPv6 sont définis dans le module des en-têtes d'extension (EXTHDRS). Quand un paquet MIPv6 rentre dans un nœud, « MIPGlue », qui fait le lien entre IPv6 et MIPv6 est appelé (étape 1). Ensuite, lors de l'étape 2, le paquet est traité. On l'envoie aussi au module DESTOPTS pour traiter les options de destination (étape 3). Lors de l'étape 4, on vérifie la durée de vie du BU et on envoie un « Proxy Neighbor Advertisement » si le nœud mobile n'est pas sur le réseau d'origine. La « Binding Cache » est ensuite mise à jour à l'étape 5. On met également à jour la table de routage et on crée le tunnel pour le nœud mobile. Ensuite, on envoie l'acquittement (étape 6). Pour pouvoir l'envoyer, on doit le construire correctement (étape 7). Une fois ceci effectué, on peut l'envoyer (étape 8).

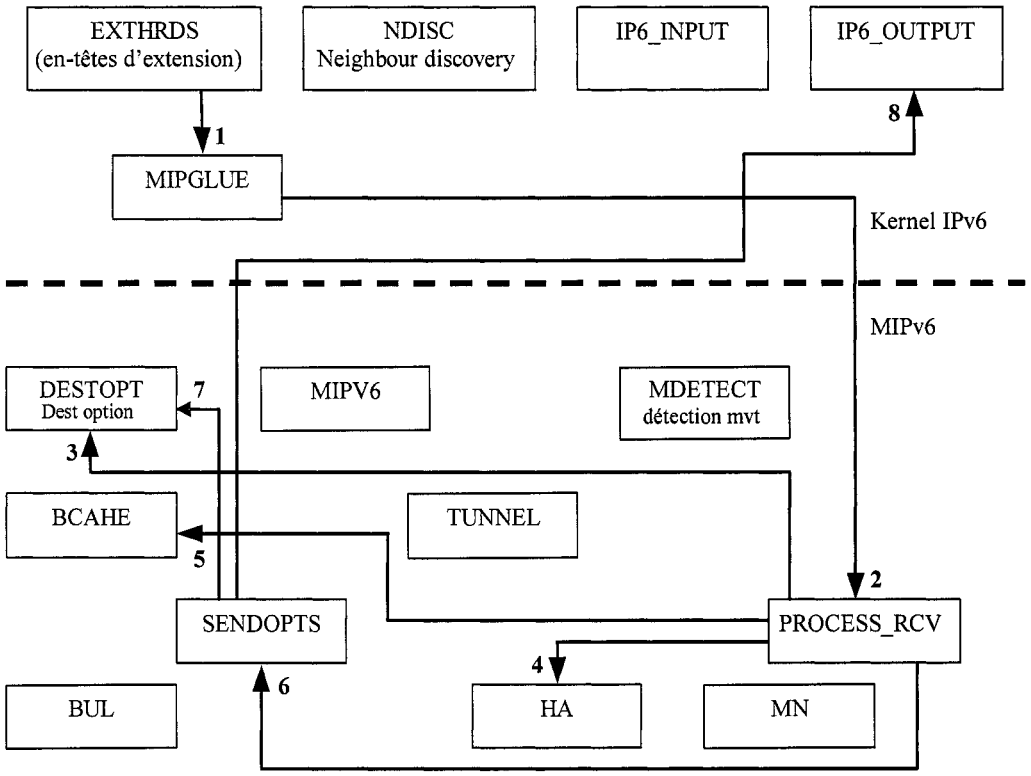


Figure 4.5 HA reçoit un BU

La Figure 4.6 illustre le cas où un HA intercepte un paquet et l'envoie par le tunnel au nœud mobile. Quand un paquet arrive, il est traité par le HA (étape 1). Le HA vérifie s'il existe en « Binding Cache » une association d'adresse pour l'adresse de destination spécifié dans le paquet entrant (étape 2). Si c'est le cas, on encapsule le paquet dans une nouvelle en-tête IPv6 et on envoie le paquet.

Une fois que l'on a vu comment les paquets MIPv6 sont traités, nous pouvons commencer à penser à la séparation CE-FE en vue de pouvoir effectuer l'implémentation de l'API.

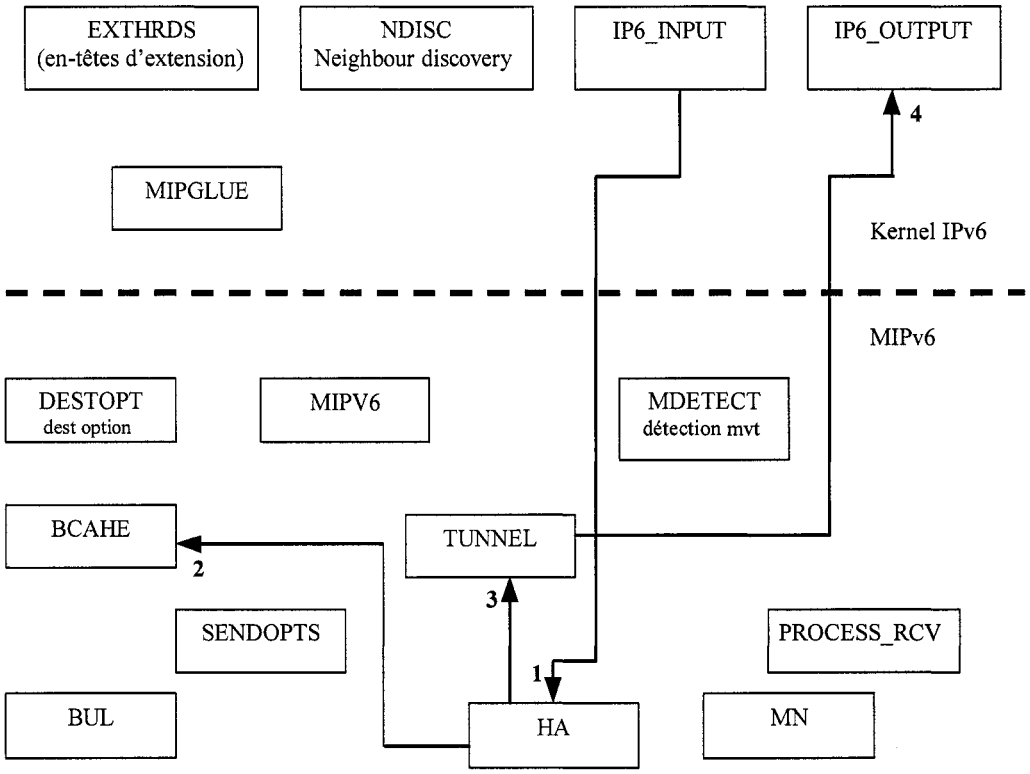


Figure 4.6 Interception d'un paquet par un HA

4.1.4 Détails supplémentaires

Notre but ici est de faire une séparation entre un CE et un FE pour ressembler à ce qui se ferait dans un véritable système. Le test de l'API nécessite la création d'une partie contrôle et d'une partie transmission.

La partie transmission a plusieurs opérations à effectuer :

1. Recevoir les paquets entrant, déterminer s'ils sont destinés ou non à cet hôte et le transmettre à la partie contrôle ;
2. Recevoir des messages en provenance du CE pour la mise à jour du FIB ;
3. Envoyer un acquittement au CE.

La partie contrôle s'occupe de la gestion des paquets MIPv6 et de la communication avec le FE.

Le code MIPL s'insérant à l'intérieur du noyau, il est beaucoup plus simple d'implémenter les parties CE et FE en tant que modules du noyau. Ceux-ci pourraient être codés au niveau utilisateur. Toutefois, séparer le code artificiellement en niveau noyau et utilisateur nécessiterait l'implémentation d'une nouvelle interface qui requerrait beaucoup de travail supplémentaire.

Comme on le voit, une communication doit se faire entre les composantes CE et FE. Pour cela, nous avons choisi TIPC (« Telecom Inter Process Communication ») [24] qui permet une communication inter-processus fiable et rapide. TIPC, qui a été développé à Ericsson, fournit une qualité de service garantissant l'absence de perte de paquets et la non duplication des messages. Il se présente sous la forme d'un module chargeable dans le noyau linux. TIPC est simple d'utilisation car son interface de programmation est très proche de celui des sockets BSD communément utilisés. La Figure 4.7 montre la structure interne de TIPC.

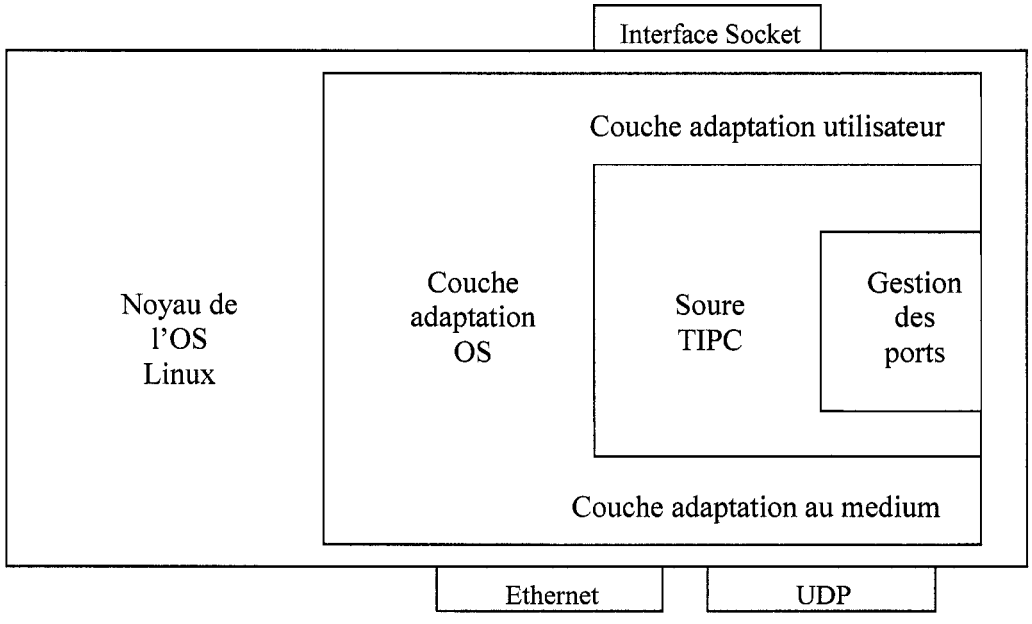


Figure 4.7 Structure interne de TIPC

Les différents éléments nécessaires à l'implémentation de l'API étant identifiés, nous pouvons maintenant étudier en détail sa structure.

4.2 Implémentation

Dans cette partie, nous allons exposer l'implémentation de l'API. Nous décrirons tout d'abord les modifications nécessaires au code MIPL de manière à pouvoir le séparer en une partie CE et une partie FE, ensuite nous analyserons les interactions entre ces deux parties pour la mise en œuvre de l'API.

4.2.1 Modification à MIPL

Comme nous l'avons déjà dit, la partie CE s'occupe du contrôle des paquets tandis que la partie FE s'occupe de l'acheminement des paquets. Dans cette optique, nous voyons que la grosse partie du travail est effectuée par la partie contrôle. Lorsqu'un paquet arrive sur une interface d'entrée, on analyse en fonction du contenu de la FIB si le paquet est destiné à cet hôte ou non. S'il n'est pas destiné à cet hôte, le paquet ne dépasse pas le FE et est tout de suite acheminé vers sur une interface de sortie. Si le paquet est destiné à cet hôte, il est envoyé au CE qui s'occupe de l'analyse du paquet. Si ce paquet est un paquet de gestion, il est possible que le CE demande au FE une mise à jour de la FIB. Comme on peut le constater, le FE ne prend aucune décision. Son comportement est dicté par le contenu de la FIB et par les messages que lui envoie le CE. Ceci est nécessaire, car l'on veut que le FE n'ait pas beaucoup de travail à faire de manière à ce que les paquets puissent être transmis le plus vite possible. De son coté, le CE a beaucoup de travail, il reçoit les paquets du FE, les analyse et, le cas échéant, retourne un message de mise à jour au FE.

Cette séparation de la charge de travail entre le CE et le FE signifie que la plus grosse partie du code MIPL se retrouve du coté du CE. Par exemple, quand un message de mise à jour d'association d'adresse (BU) arrive sur une interface, on regarde à l'intérieur de la table de routage si ce paquet a pour destination finale cet

hôte. Si c'est le cas, le FE transmet le paquet au CE. Le CE regarde le paquet et constate qu'il s'agit d'un paquet de mise à jour d'association d'adresse. À ce moment, il appelle la fonction de traitement de ce genre de paquet défini dans MIPL. Si ce paquet est valide, suivant les spécifications définies par le protocole MIPv6, le CE doit demander au FE la création d'un tunnel entre l'adresse du HA et la CoA du MN. Si cette opération effectuée au niveau du FE réussit, celui-ci doit également créer une route pour l'adresse nominale du MN à travers ce tunnel. Le FE n'engage pas ce travail quand il reçoit un BU mais uniquement à la demande du CE. Comme nous le montre ce petit exemple, les appels de fonction du code MIPL ont été déplacés de manière à rendre cohérentes les parties CE et FE. Les fonctions de traitement des paquets définis dans MIPL sont appelées à partir du CE.

4.2.2 Mise en œuvre de l'API

L'exemple précédent nous montre bien qu'il y a de nombreuses et très importantes communications entre le FE et le CE. C'est ici qu'entrent en jeu les communications utilisant le protocole TIPC. Les FE et CE sont en fait des serveurs TIPC qui sont capables de recevoir et d'envoyer des messages permettant l'implémentation de l'API. On y retrouve sept étapes principales décrites à la Figure 4.8:

1. un paquet arrive sur une interface entrante du FE ;
2. le paquet est alors transmis au CE ;
3. le CE appelle les fonctions MIPL pour le traitement du paquet ;
4. en fonction du contenu du paquet le CE demande au FE, par l'intermédiaire des fonctions de l'API, la mise à jour de la FIB ;
5. le FE exécute les commandes du CE et fait ses mises à jour ;
6. Suivant le résultat de l'opération effectuée, le FE retourne une réponse positive ou négative au CE ;
7. Finalement, le CE peut mettre à jour la RIB ou demander des corrections au FE si l'opération a échoué.

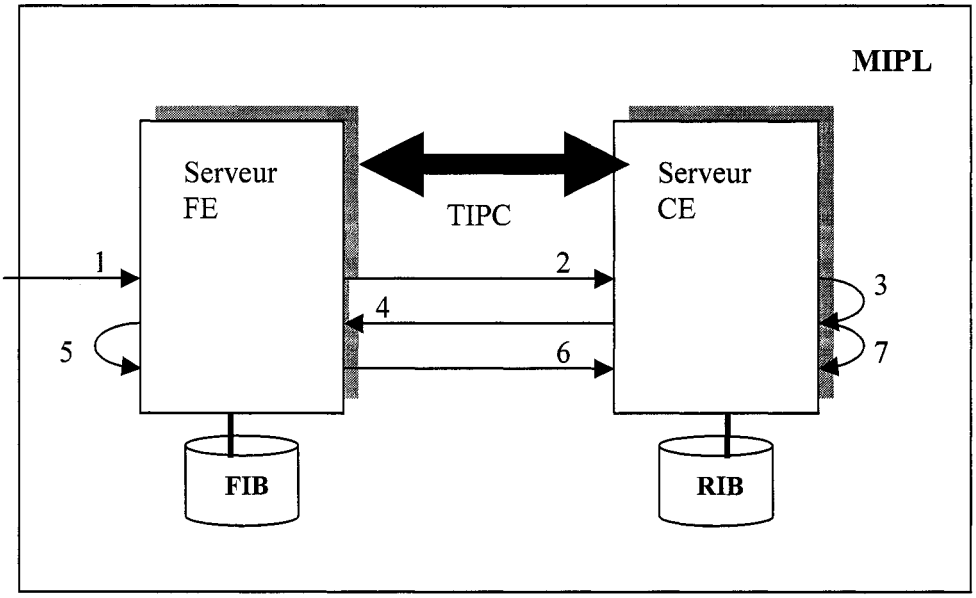


Figure 4.8 Architecture API

Le CE doit se souvenir des dernières opérations effectuées au niveau du FE. Si par exemple nous recevons un message de mise à jour d'association d'adresse, le CE doit demander au FE de créer un tunnel et d'ajouter une route pour le nœud mobile. Ensuite, nous mettons à jour la table d'association d'adresse. Si cette dernière opération échoue, le tunnel et la route créés auparavant n'ont plus aucune raison d'être. Si on ne peut pas mettre à jour la table d'association d'adresse, le CE doit demander d'enlever le tunnel et la route.

4.3 Plan d'expérience

Le plan d'expérience vise à vérifier si l'API et son implémentation fonctionnent. On ne va pas chercher à faire des tests de performance, car de toute façon, puisque l'API n'est pas destinée à une implémentation toute « software », et puisque nous avons utilisé le code MIPL en y ajoutant l'implémentation de l'API ainsi que des messages TIPC supplémentaires, le temps de traitement ne peut qu'en être augmenté. Nous ne cherchons dans cette section qu'à faire un test fonctionnel de l'API, en montrant quelques exemples significatifs.

4.3.1 Description du réseau

Le réseau utilisé pour tester l'implémentation de l'API est présenté à la Figure 4.9.

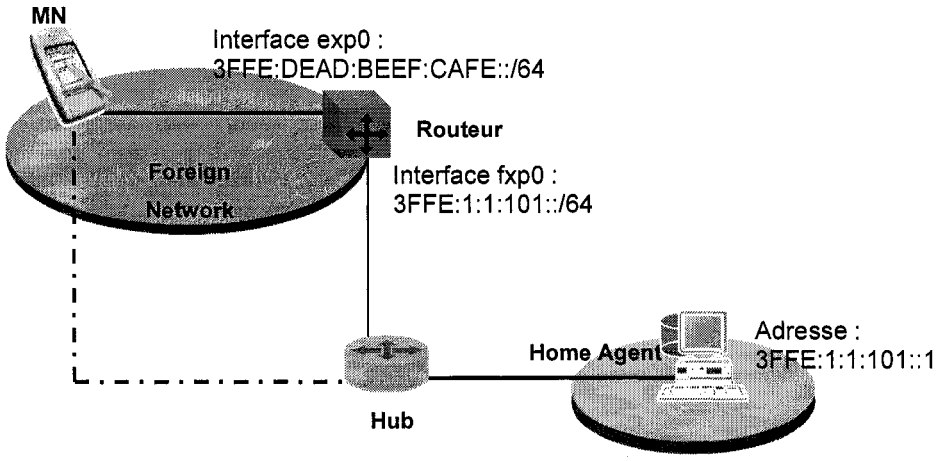


Figure 4.9 Réseau de test

Ce réseau n'est pas un réseau sans fil. On déplace le MN en le branchant sur un réseau différent. Les relèves sont donc très abruptes car il n'y a pas de chevauchement de cellules. Le réseau est constitué de trois nœuds importants :

1. Le HA situé sur le réseau d'origine possède l'adresse 3ffe:1:1:101::1. Pour que le MN reconnaisse qu'il a rejoint ce réseau, le HA doit envoyer des RA, ainsi RADVD (« Router Advertisement daemon ») y a été configuré ;
2. Le MN est sur le réseau d'origine et quand il se déplace sur l'un ou l'autre des deux réseaux, il est obligé de générer une adresse CoA ;
3. Le routeur fait le lien entre le réseau d'origine et le réseau visité par le MN. Il possède deux interfaces. La première (fxp0) a une adresse basée sur le préfixe du réseau d'origine (3FFE:1:1:101::/64). La seconde est reliée au réseau visité et annonce le préfixe de ce réseau (3FFE:DEAD:BEEF:CAFE::/64) à partir duquel le MN devra construire son adresse.

4.4 Évaluation

Au tout début de la simulation, si le nœud mobile se trouve sur le réseau d'origine, celui-ci reçoit un RA indiquant que le préfixe du réseau où il se trouve correspond au préfixe utilisé pour l'adresse du HA. Cela signifie que le MN est sur le même réseau que le HA. Le MN n'a donc pas à envoyer de BU au HA. Dans ce cas de figure, le HA n'a pas à intercepter les paquets pour le nœud mobile ; le MN les recevra lui-même. Les Figures 4.10 et 4.11 montrent ce qui se passe au niveau du CE et du FE du HA. (Ces figures, ainsi que les suivantes, sont des portions de figures tirées des résultats de l'implémentation. Les figures ne sont par montrées au complet ici pour garantir la lisibilité. Les vraies figures se retrouvent en annexe.)

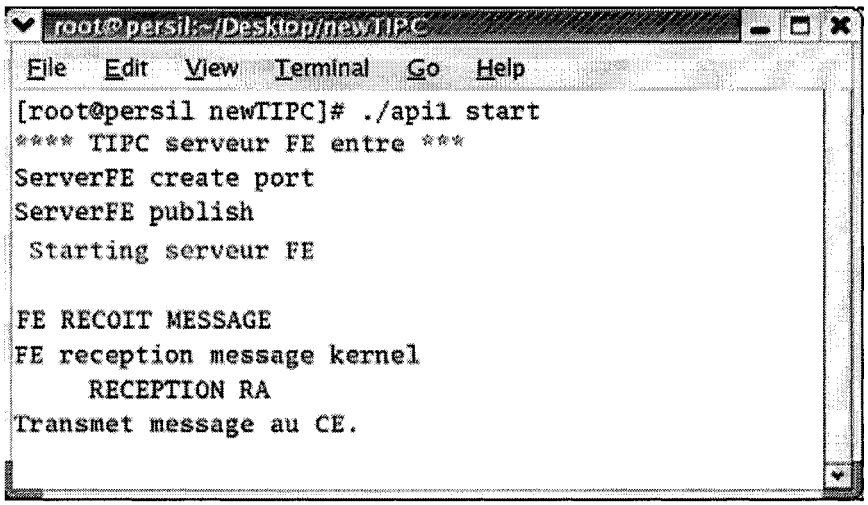


Figure 4.10 Réception d'un RA par le HA

Le RA envoyé par le deamon RADVD est intercepté par le FE qui le transmet directement au niveau contrôle. Le CE reçoit ce message et l'analyse. L'adresse lien local (ll_addr) de l'interface réceptrice, dont l'index est 3, est FE80:0:0:0:208:C7FF:FE85:C9AF. On peut également voir que le préfixe du HA annoncé est bien le 3FFE:1:1:101::/64. Quand le MN reçoit ce message, il remarque qu'il est sur le réseau du HA et c'est la raison pour laquelle aucun BU n'est envoyé.

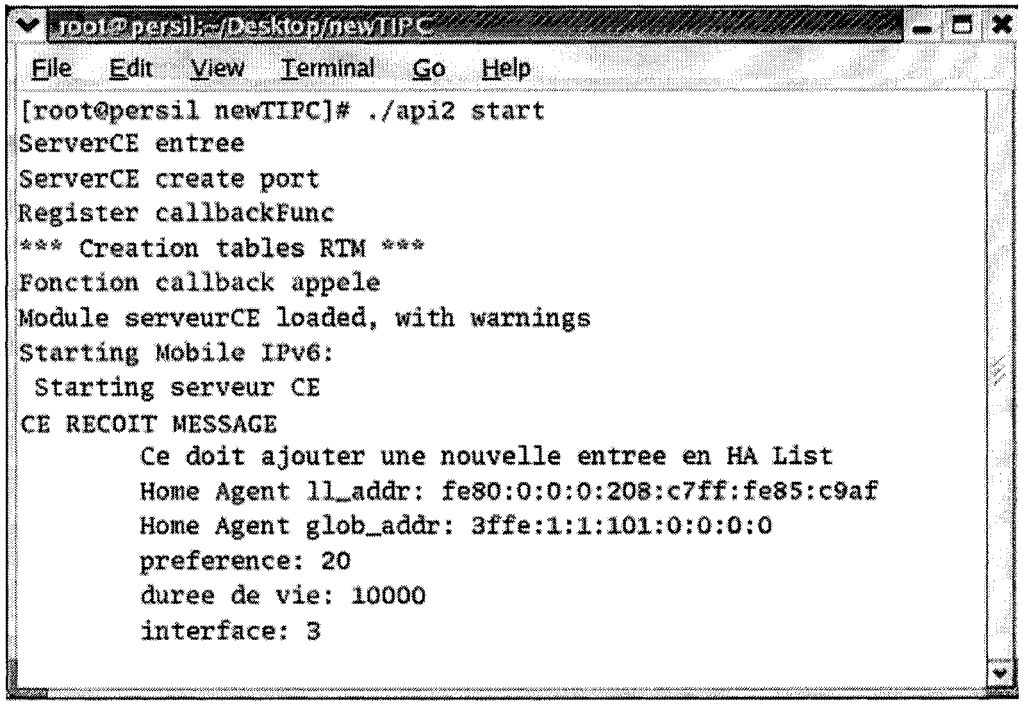


Figure 4.11 Traitement du RA

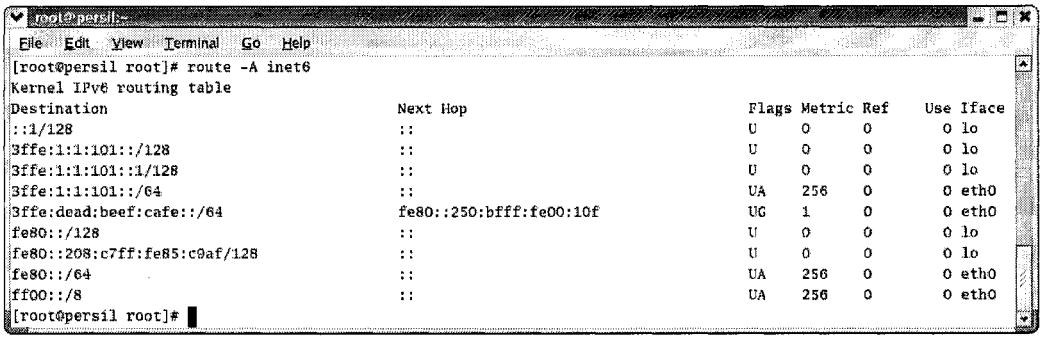


Figure 4.12 Table de routage du noyau

La Figure 4.12 montre le contenu de la table de routage du noyau du HA. On voit simplement que l'on peut rejoindre le réseau distant à partir de l'interface eth0 du HA.

Si on déplace le MN sur un autre réseau, il utilise le mécanisme de détection de mouvement grâce à la découverte de voisins. Cela permet de déterminer si le routeur par défaut est encore rejoignable. Si l'ancien routeur n'est plus rejoignable, le MN doit trouver un nouveau routeur par défaut. Le MN sur un autre réseau doit

envoyer un message « Router Solicitation » multicast. Le AR appelé routeur dans notre réseau test va répondre avec son propre préfixe. Le MN peut alors construire sa nouvelle adresse IPv6 en fonction du préfixe reçu et de son adresse MAC. À ce moment, le MN envoie un « Binding Update » au HA. Les Figures 4.13 et 4.14 montrent ce qui se passe aux niveaux FE et CE du HA.

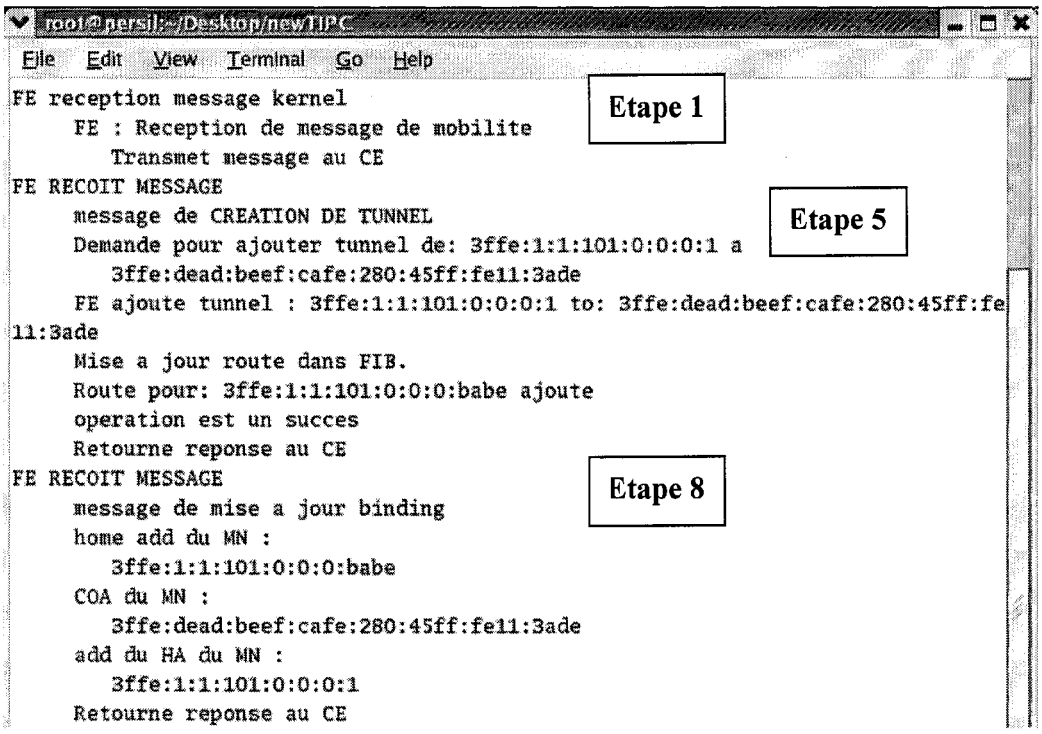


Figure 4.13 Création de tunnel et mise à jour BC au FE

Quand le HA reçoit un BU, le FE l'intercepte (étape 1). Il reconnaît qu'il s'agit d'un message de gestion de mobilité sans savoir que c'est un BU et il envoie le paquet au CE. Le CE, à l'étape 2 (Figure 4.14), regarde le type du message et reconnaît qu'il s'agit d'un « Binding Update ». À ce moment là, il appelle la fonction de traitement des BU. Si ce BU est conforme aux spécifications définies par le protocole MIPv6, la fonction de traitement final des BU est appelée (étape 3). Le CE fait alors plusieurs vérifications :

1. s'il s'agit d'une « Home Registration » (création de tunnel) ;
    a. on vérifie si l'entrée en « binding cache » (BC) existe déjà :

i. si oui, on vérifie si le tunnel existe, si oui :
    1. on efface le tunnel ;
    2. on efface la route.
ii. Si non, on crée le tunnel, si la création réussit :
    1. on crée la route, si cette création réussit :
        a. on met à jour le BC.
    2. sinon :
        a. erreur.

2. sinon :
    a. on met seulement à jour le BC uniquement (sans faire de tunnel).

Ainsi, on voit qu'à l'étape 4, le CE demande au FE la création du tunnel puisqu'il s'agit du premier BU reçu et non d'une mise à jour d'une entrée en « Binding cache ». À l'étape 5, on voit que le FE a reçu le message de création de tunnel. Il ajoute un tunnel qui a pour adresse source l'adresse du HA (3FFE:1:1:101::1) et pour adresse de destination, la CoA du MN (3FFE:DEAD:BEEF:CAFE:280:45FF:FE11:3ADE). Si la création de ce tunnel est un succès, le FE crée une route pour l'adresse nominale du MN (3FFE:1:1:101::BABE) par le tunnel créé. À ce moment, le FE retourne la réponse concernant l'opération effectuée. Le CE reçoit cette réponse à l'étape 7. À l'étape 6, le CE avait demandé la mise à jour de la BC au FE. Le FE reçoit ce message à l'étape 8 et met à jour sa cache locale. Quand le CE reçoit la réponse du FE, à l'étape 9, il peut finalement mettre à jour sa BC.

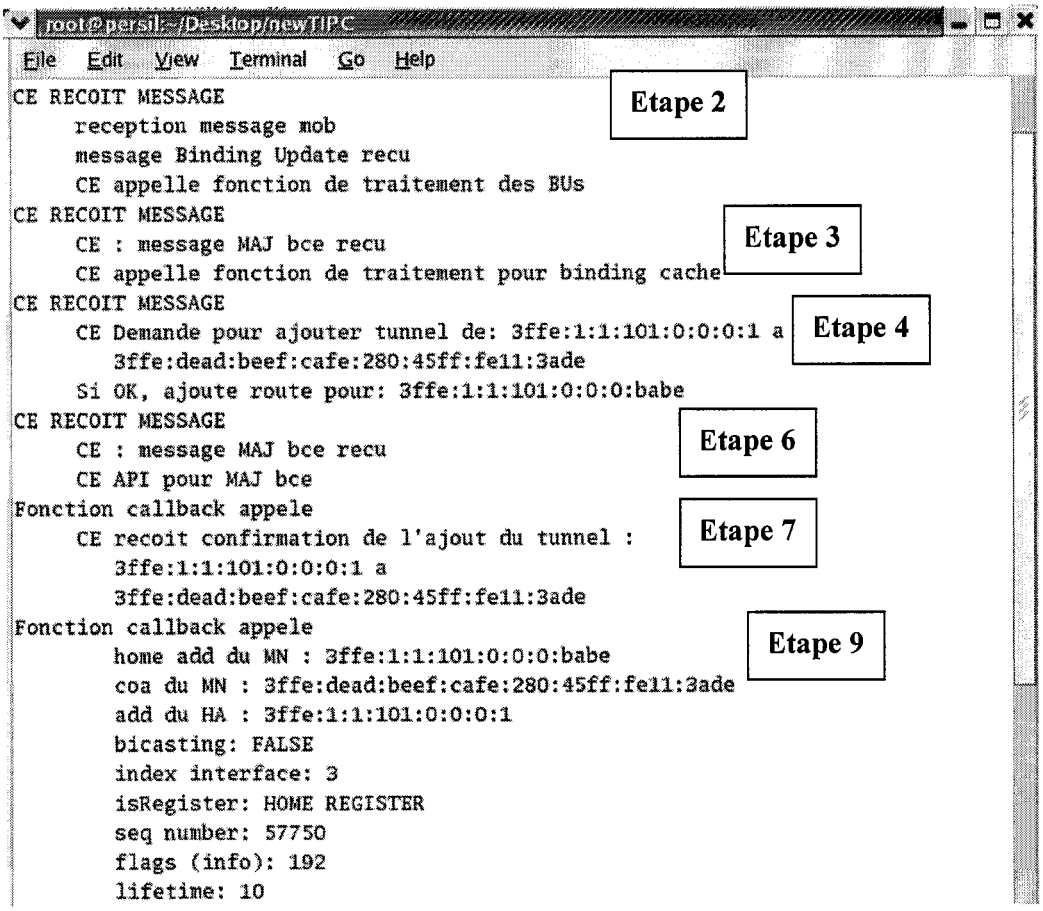


Figure 4.14 Création de tunnel et mise à jour BC au CE

On voit qu'à partir de la réception d'un seul message, plusieurs opérations doivent être effectuées. Si, par exemple, la mise à jour de la BC échoue, on doit effacer le tunnel créé. On vérifie alors si le tunnel existe, on demande au FE de l'effacer et d'effacer la route pour l'adresse nominale du nœud mobile.

Par opposition à la Figure 4.12, la Figure 4.15 montre la table de routage après la mise à jour du BU. On voit que le nœud mobile (3FFE:1:1:101::babe) est maintenant rejoignable grâce au tunnel (ip6tnl1). Le tunnel permet de faire croire qu'il n'y qu'un seul saut entre le MN et le HA. Ainsi, la « home address » du MN est basée sur le préfixe du HA. C'est pour cela que le prochain saut n'est pas indiqué. Le tunnel représente l'interface de sortie.

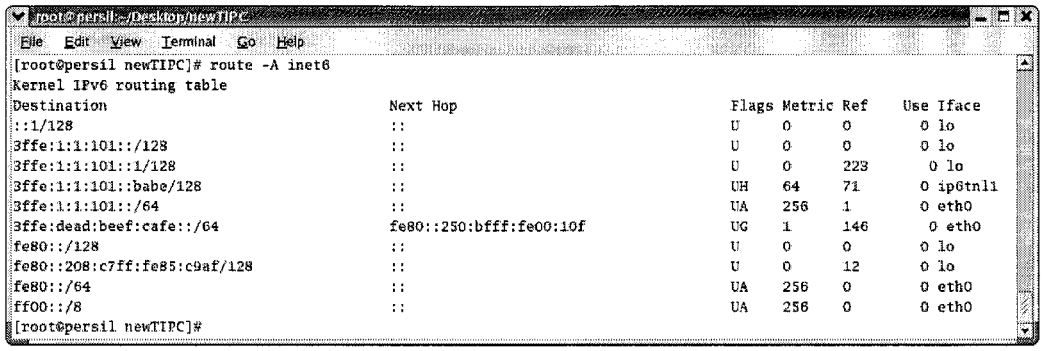


Figure 4.15 table routage noyau

Lorsque le MN retourne au réseau d'origine, il envoie un BU pour demander au HA d'arrêter d'intercepter les paquets pour le MN et de les transmettre par le tunnel. Les Figures 4.16 et 4.17 illustrent ce qui se passe au niveau du HA. Tout d'abord, le FE reçoit le paquet comme d'habitude et il le transmet au CE.

```
CE RECOIT MESSAGE
     CE : message MAJ bce recu
     Appelle API pour effacer l'entree
        Une entree en bce pour cette home add existe, on l'efface
        Un tunnel pour cette home add existe, on l'efface
Fonction callback appele
     CE recoit confirmation de l'effacement du tunnel :
        3ffe:1:1:101:0:0:0:1 a
        3ffe:dead:beef:cafe:280:45ff:fe11:3ade
```

Figure 4.16 CE Retour du MN sur le réseau d'origine

Lorsque le CE reçoit ce paquet, il se rend compte qu'il contient une demande pour effacer une association d'adresse. Après avoir vérifié dans les tables locales que cette entrée d'association existe, et qu'un tunnel existe également, il demande au FE de l'enlever. Alors, le FE enlève le tunnel et la route et retourne la réponse au CE. On voit bien à la Figure 4.16 que le CE reçoit une confirmation de l'effacement du tunnel.

```
FE RECOIT MESSAGE
     demande pour effacer tunnel
     tunnel et route effacer
     Retourne reponse au CE
```

Figure 4.17 FE Retour du MN sur le réseau d'origine

Cette implémentation ne couvre pas l'API au complet. Elle a été conçue pour couvrir la suite des protocoles de gestion de la mobilité. Or, MIPL n'implémente que MIPv6. Mais puisque le draft issu de l'API sera présenté à l'IETF (« Internet Engineering Task Force », http://www.ietf.org), des échos supplémentaires pourront nous revenir. Cette implémentation sert surtout à valider la base de l'API. Les résultats que nous avons présentés ici sont conformes aux spécifications de « FORCES Working Group » (http://www.ietf.org/html.charters/forces-charter.html) et confirment la validité fonctionnelle de l'API.

CHAPITRE V

CONCLUSION

En guise de conclusion de notre mémoire, nous allons synthétiser notre proposition et en décrire les limitations. Enfin, nous proposerons une série de travaux futurs possibles.

5.1 Synthèse de la proposition

Notre travail s'est basé sur la séparation des routeurs en parties CE et FE. La partie FE est une partie matérielle. Cette partie comprend plusieurs cartes comprenant chacune un ou plusieurs processeurs réseaux et comprenant une ou plusieurs FIB. La partie CE est de nature plus logicielle et s'occupe du traitement interne des paquets. Un FE comprend un RIB qui est la table de routage du routeur. Une communication entre ces deux parties est essentielle pour assurer une cohérence entre FIB et RIB. C'est dans ce contexte qu'une API entre ces deux parties a été définie. L'API que nous avons définie ne s'occupe pas de l'association entre les CE et FE, mais uniquement de la gestion de la mobilité.

Les protocoles définis par l'IETF deviendront bientôt des standards. Par contre, la capacité des processeurs réseaux s'accroît très rapidement. Pour disposer d'un routeur efficace, le CE ne devrait pas beaucoup changer tandis que le changement des FE peut être régulier. L'API faisant le lien entre les deux parties se doit de rester constant.

L'implémentation que nous avons faite ne s'occupe pas de grappe d'ordinateurs car l'API est le même entre toutes les partie CE et FE. De plus, les phases de pré et de post-association entre CE et FE à l'intérieur d'une grappe d'ordinateurs ont déjà été traitées par Rachid Nait Takourout lors de sa maîtrise au LARIM [25].

Pour valider notre API, nous avons testé un cas simple mais représentatif des échanges de messages de gestion de la mobilité. Les résultats obtenus montrent que l'API et son implémentation fonctionnent correctement. De plus, deux documents seront proposés à l'IETF, le premier traite de l'API de gestion de mobilité [21] et le second traite de l'API pour la gestion des interfaces [22]. La présentation de ces deux documents appuyés par l'implémentation que nous avons faite nous permettra de recueillir des commentaires supplémentaires.

5.2 Limitations de notre proposition

La limitation principale dans notre travail, c'est que l'API n'a pas été implémenté au complet. Il y a deux raisons principales à cette restriction : premièrement, MIPL ne couvre pas toute la suite des protocoles de gestion de mobilité, et secondement, notre réseau de test ne couvre pas tous les cas possibles dans l'échange des messages de gestion. De plus, l'implémentation de l'API n'a pas été réalisée sur un routeur pour lequel il est dédié. On a fait notre implémentation sur un routeur tout logiciel. Ainsi, notre FE est logiciel au lieu d'être matériel. Ceci fait que notre implémentation est plus lente car, de plus, on ajoute à tout le code présent dans MIPL des messages TIPC supplémentaires nécessaires à la communication entre le FE et le CE.

5.3 Travaux futurs

Tout d'abord, il serait pertinent d'intégrer notre implémentation au sein d'un véritable routeur. L'étude des performances que l'on pourrait y faire permettrait de connaître précisément l'impact de l'API. Cela nous permettrait d'analyser la performance et de voir réellement si cette solution permet de supporter du trafic en

temps réel. Mais, pour réaliser une telle intégration, il faudrait créer de nouveaux messages pour le support de la gestion de la mobilité suivant le protocole FACT avec XML, comme l'a fait Rachid Nait Takourout dans son étude de la modélisation des données des phases de pré et de post-association.

Par ailleurs, il serait pertinent d'implémenter l'API au complet. Cependant, la sortie d'un code du genre MIPL couvrant toute la suite des protocoles de gestion de mobilité avec MIPv6 n'est encore à l'ordre du jour. Ceci devrait pouvoir se réaliser dans un proche avenir.

Finalement, pour compléter le tout et avoir un système fonctionnel, il faudrait également proposer et implémenter d'autres API comme par exemple, ceux couvrant les aspects de sécurité comme IPSec.

BIBLIOGRAPHIE

[1] D. Johnson, D. Perkins, J. Arkko, « Mobility Support in IPv6 », draft-ietf-mobileip-ipv6-24.txt, June 2003 Work in Progress.

[2] H. Soliman, C. Castelluccia, K. El-Malki, L. Bellier, « Hierarchical Mobile IPv6 mobility management (HMIPv6) », draft-ietf-mipshop-hmipv6-01.txt, Octobre 2004

[3] R. Koodli, « Fast Handovers for Mobile IPv6 », draft-ietf-mobileip-fast-mipv6-06, 30 January 2004.

[4] H. Young Jung, S. J. Koh, H. Soliman, K. El-Malki, B. Hartwell « Fast Handover for Hierarchical MIPv6 (F-HMIPv6) <draft-jung-mobileip-fastho-hmipv6-03.txt> », Juin 2004

[5] K. ElMalki, H. Soliman, « Simultaneous Bindings for Mobile IPv6 Fast Handoffs » , draft-elmalki-mobileip-bicasting-v6-05.txt, Oct 2003.

[6] N. Moore, « Optimistic Duplicate Address Detection », draft-moore-ipv6-optimistic-dad-02, February 2003.

[7] H. Jung, « Address Pool based Stateful NCoA Configuration for FMIPv6 », draft-jung-mipshop-stateful-fmipv6-00, August 2003.

[8] A. Conta, S. Deering, « Generic Packet Tunneling in IPv6 Specification » RFC 2473.

[9] T. Narten, E. Nordmark, W. Simpson, « Neighbor Discovery for IP Version 6 (IPv6) » , RFC 2461, December 1998.

[10] L. Yang, R. Dantu « Requirement for Separation of IP Control and Forwarding » draft-ietf-forces-framework-13.txt, January 2004

[11] L.Yang, J. Halpern, R. Gopal, A. DeKok, Z. Haraszti, S. Blake, E. Deleganes « ForCES Forwarding Element Model », draft-ietf-forces-model-02.txt, February 2004

[12] L. Yang, « Forwarding and Control Element Separation (ForCES) Framework », draft-ietf-forces-framework-13.txt, January 2004

[13] Intel ® « Control Plane Platform Development Kit, Software Architecture Overview », Février 2003

[14] D. Plummer, « An Ethernet Address Resolution Protocol -or-Converting Network Addresses to 48-bit Ethernet Address forTransmission on Ethernet Hardware », STD 37, RFC 826, November 1982.

[15] K.Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith, « COPS Usage for Policy Provisionning (COPS_PR) » , RFC3084, Mars 2001.

[16] C. Doria, « GSMPv3 Base Specification », draft-ietf-gsmp-v3-base-spec-05, November 2003

[17] The Object Management Group, « Common Object Request Broker Architecture Specification 2.2 », http://www.omg.org

[18] K. Gregersen, « IPv6 Unicast Forwarding Service API Implementation Agreement », NPF2002.616.05, 2002.

[19] J. Moy, « OSPF Version 2 » , RFC 1583, March 1994.

[20] Y. Rekhter, T. Li, « A Border Gateway Protocol 4 (BGP-4) », RFC 1771, March 1995.

[21] V. Passelande, « MIPv6 API Specification », 2003

[22] V. Passelande, « NPF MIPv6 Interface Management API », 2003

[23] A. Kulkarni, « IPv6 Forwarding API Specification », NPF2002.382, July 17, 2002

[24] J. Maloy, TIPC, http://tipc.sourceforge.net

[25] R. Nait Takourout, « Étude de la séparation entre la partie contrôle et la partie acheminement dans les routeurs », mémoire de maîtrise (M.Sc.A), École Polytechnique de Montréal, septembre 2003

ANNEXE A

API

```
#ifndef __NPF_H__
#define __NPF_H__

#ifdef __cplusplus
extern "C" {
#endif

//#include <linux/in6.h>
//#include <net/ipv6.h>

#define   NPF_IN
#define   NPF_OUT
#define   NPF_IN_OUT

/* This section defines base NPF types and will differ from */
/* platform to platform. The type shown here are based on */
/* Linux 6.2 on an x86. */

typedef    char                          NPF_char8_t;
typedef    unsigned char                 NPF_uchar8_t;
typedef    char                          NPF_int8_t;
typedef    short                         NPF_int16_t;
typedef    int                           NPF_int32_t;
typedef    long long int                 NPF_int64_t;
typedef    unsigned char                 NPF_uint8_t;
typedef    unsigned short                NPF_uint16_t;
typedef    unsigned int                  NPF_uint32_t;
typedef    unsigned long long int        NPF_uint64_t;
typedef    float                         NPF_float32_t;
typedef    long double                   NPF_float64_t;

/* This section defines constructed NPF types and is */
/* identical for all implementations of the NPF APIs. */
typedef    NPF_uint32_t      NPF_error_t;
typedef    NPF_uint32_t      NPF_callbackHandle_t;
typedef    NPF_uint32_t      NPF_correlator_t;
typedef    NPF_uint32_t      NPF_userContext_t;
typedef    enum NPF_boolean {NPF_FALSE=0,NPF_TRUE = 1}NPF_boolean_t;
typedef    NPF_uint32_t      NPF_IPv4Address_t;
typedef    struct in6_addr   NPF_IPv6Address_t;
typedef    NPF_uchar8_t      NPF_MAC_Address_t[6];

typedef    enum NPF_errorReporting {
        NPF_REPORT_ALL = 1,
        NPF_REPORT_NONE = 2,
        NPF_REPORT_ERRORS = 3
} NPF_errorReporting_t;

#define NPF_NO_ERROR 0
```

```
#define NPF_FOUNDATIONS_BASE_ERR 1
#define NPF_FOUNDATIONS_MAX_ERR (NPF_FOUNDATIONS_BASE_ERR + 98)

#define NPF_E_UNKNOWN                          NPF_FOUNDATIONS_BASE_ERR
#define NPF_E_BAD_CALLBACK_HANDLE             (NPF_FOUNDATIONS_BASE_ERR
                                               + 1)
#define NPF_E_BAD_CALLBACK_FUNCTION           (NPF_FOUNDATIONS_BASE_ERR
                                               + 2)
#define NPF_E_CALLBACK_ALREADY_REGISTERED     (NPF_FOUNDATIONS_BASE_ERR
                                               + 3)

#ifdef __cplusplus
}
#endif

#endif  /* __NPF_H__ */
```

MIPv6 API

```
#ifndef NPFMIPV6API_H
#define NPFMIPV6API_H

#ifdef __cplusplus
extern "C" {
#endif

/*************** MOBILITY HEADER CONSTANTS ********************/
#define MH_TYPE_BRR          0  /* Binding Request */
#define MH_TYPE_HOTI         1  /* HOTI Message  */
#define MH_TYPE_COTI         2  /* COTI Message  */
#define MH_TYPE_HOT          3  /* HOT Message  */
#define MH_TYPE_COT          4  /* COT Message  */
#define MH_TYPE_BU           5  /* Binding Update */
#define MH_TYPE_BACK         6  /* Binding ACK */
#define MH_TYPE_BERROR       7  /* Binding Error */

/********** MOBILITY HEADER MESSAGE OPTION TYPES *************/
#define  MHOPT_PAD1          0x00  /* PAD1 */
#define  MHOPT_PDAN          0x01  /* PADN */
#define  MHOPT_UID           0x02  /* Unique ID */
#define  MHOPT_ALTCOA        0x03  /* Alternate COA */
#define  MHOPT_NONCEID       0x04  /* Nonce Index */
#define  MHOPT_BAUTH         0x05  /* Binding Auth Data */
#define  MHOPT_BREFRESH      0x06  /* Binding Refresh */

      /* STATUS VALUES ACCOMPANIED WITH MOBILITY BINDING
                          ACKNOWLEDGEMENT */
#define MH_BAS_ACCPETED          0  /* Binding update accepted */
#define MH_BAS_UNSPECIFIED       128 /* Reason unspecified */
```

```
#define MH_BAS_ADMIN                129 /* Administratively prohibited */
#define MH_BAS_INSUFFICIENT         130 /* Insufficient resources */
#define MH_BAS_NOT_HA       131 /* HA registration not supported */
#define MH_BAS_NOT_HOME_SUBNET132 /* Not Home subnet */
#define MH_BAS_WRONG_HA             133 /* Not HA for this mobile node */
#define MH_BAS_DAD_FAILED           134 /* DAD failed */
#define MH_BAS_SEQNO_BAD            135 /* Sequence number out of range */
#define MH_BAS_EXP_HOME_NI          136 /* Expired Home nonce index */
#define MH_BAS_EXP_COA_NI           137 /* Expired Care-of nonce index */
#define MH_BAS_EXP_NI               138 /* Expired Nonce Indices */

/************************ Router Advertisement ************************/
#define MAXMOBPFXADVINTERVAL  86400  //seconds
#define MINMOBPFXADVINTERVAL  600    //seconds
#define PREFIX_ADV_TIMEOUT    3      //seconds
#define PREFIXE_ADV_RETRIES   3      //retransmission

typedef NPF_uint32_t        NPF_timer32_t;


/****************************** Binding ******************************/
typedef struct{
        NPF_IPv6NetAddress_t        mn_home_addr;
        NPF_IPv6NetAddress_t        mn_co_addr;
        NPF_IPv6NetAddress_t        mn_ha_addr;
        NPF_boolean                 bicasting;
        NPF_IFGeneric_t             interfaceHandle;
        NPF_boolean                 isRegister;
        NPF_uint16_t                seqNumber;
        NPF_uint8_t                 info;
        NPF_timer32_t               lifetime;
        NPF_uint32_t                tunnelID;
        NPF_IPv6UnicastForwardingTableHandle_t FIB_Handle
} NPF_MIPv6BindingEntry_t;

typedef struct{
        NPF_uint32_t                bindingCount;
        NPF_MIPv6BindingEntry_t  *bindingEntryArray;
}NPF_MIPv6BindingEntryArray_t;


typedef struct {
        NPF_uint32_t                tunnelID;
        NPF_IfHandle_t              interfaceHandle;
        NPF_IPv6NetAddress_t        tunnel_IPv6SrcAddr;
        NPF_IPv6NetAddress_t        tunnel_IPv6DstAddr;
        NPF_IPv6NetAddress_t        home_add;
```

```
        NPF_timer32_t           lifetime;
} NPF_ MIPv6Tunnel_t;

typedef struct {
        NPF_uint32_t            nbEntries;
        NPF_MIPv6Tunnel_t       *tunnelArray;
} NPF_MIPv6TunnelArray_t;

/***************************** Destination *****************************/
typedef struct NPF_IPv6DestEntry_s{
        NPF_IPv6NetAddress_t    IP_Address;
        NPF_IPv6_Reachability_t accessState;
        NPF_IFGeneric_t         interfaceHandle;
        NPF_MediaAddressEntry_t mediaAddress;
        NPF_boolean             isRouteur;
        NPF_uint32_t            nextHopIndex;
        NPF_uint8_t             multicastSolicitation;
        NPF_timer32_t           timer;
        NPF_uint8_t             unicastSolicitation;
} NPF_MIPv6DestEntry_t;

typedef struct{
        NPF_uint32_t            nbrCount;
        NPF_MIPv6DestEntry_t    *DestEntryArray;
} NPF_MIPv6DestEntryArray_t;

/*********************************                                    Router
*********************************/
typedef struct {
        NPF_MIPv6NeighborEntry_t        *NeighbourEntry;
        NPF_timer32_t                   retransInterval;
        NPF_timer32_t                   reachableTime;
        NPF_timer32_t                   rtrLifetime;
        NPF_boolean                     H;
} NPF_MIPv6RouterEntry_t;

typedef struct{
        NPF_uint32_t            nbEntries;
        NPF_MIPv6RouterEntry_t  *routerEntryArray;
} NPF_MIPv6RouterEntryArray_t;

/************************** HA/MAP List **************************/
typedef struct {
        NPF_uint32_t        ifindex;
        NPF_IPv6Address_t   IPAddress;
        NPF_timer32_t       lifetime;
        NPF_timer32_t       nbGlobalAdd;
```

```
        NPF_IPv6NetAddress_t    globalAddress;
        NPF_IPv6NetAddress_t    llAdd;
        NPF_uint8_t             distPref;
        NPF_uint8_t             opMode;
}NPF_MIPv6HAEntry_t;

typedef struct{
        NPF_uint32_t            nbEntries;
        NPF_MIPv6HAEntry_t      *HAEntryArray;
} NPF_MIPv6HAEntryArray_t;

/************************* Aggregate Prefix *************************/
typedef struct {
        NPF_uint32_t            nbPrefix;
        NPF_IPv6NetAddr_t       *PrefixArray;
        NPF_timer32_t           validLifetime;
        NPF_timer_t             prefLifetime;
} NPF_MIPv6PrefixAggregateEntry_t;

typedef struct {
        NPF_uint32_t            nbEntries;
        NPF_MIPv6PrefixAggregateEntry_t *prefixAggreagateEntryArray;
        NPF_timer32_t           maxMobPfxAdvInterval;
        NPF_timer32_t           maxScheduleDelay;
        NPF_timer32_t           randAdvDelay;
        NPF_timer32_t           advTimeout;
        NPF_uint8_t             advRetries;
} NPF_MIPv6PrefixAggregateEntryArray_t;

/*************************** CoA Pool ***************************/
typedef struct {
        NPF_uint8_t             nbDadFreeCoA;
        NPF_IPv6NetAddr_t *dadFreeCoa;
}NPF_AdvanceDADentry_t;

typedef struct {
        NPF_IPv6NetAddr_t       *PrefixArray;
        NPF_uint8_t             capacity_of_pool;
        NPF_AdvanceDADEntry_t   *dadFreeCoA_entryArray;
        NPF_IFGeneric_t         interfaceHandle;
}NPF_AdvanceDADEntryArray_t;

/************************ Handle of the tables ************************/
typedef NPF_uint32_t NPF_MIPv6DestTableHandle_t;
typedef NPF_uint32_t NPF_MIPv6_TunnelTableHandle_t;
typedef NPF_uint32_t NPF_MIPv6_BindingTableHandle_t;
```

```
/************************ Callback Type ***************************/
typedef enum NPF_MIPv6CallbackType {
        //Destination Cache
        NPF_MIPV6_DEST_TABLE_HANDLE_CREATE      = 1,
        NPF_MIPV6_DEST_TABLE_HANDLE_DELETE      = 2
        NPF_MIPV6_DEST_ENTRY_ADD                = 3,
        NPF_MIPV6_DEST_ENTRY_DELETE             = 4,
        NPF_MIPV6_DEST_TABLE_FLUSH              = 5,
        NPF_MIPV6_DEST_ENTRY_SET                = 6,
        NPF_MIPV6_DEST_TABLE_ATTR_QUERY         = 7,

        //Tunnel
        NPF_MIPv6_TUNNEL_TABLE_HANDLE_CREATE    = 8,
        NPF_MIPv6_TUNNEL_TABLE_HANDLE_DELETE    = 9,
        NPF_MIPv6_TUNNEL_ENTRY_ADD              = 10,
        NPF_MIPv6_TUNNEL_ENTRY_DELETE       = 11,
        NPF_MIPv6_TUNNEL_TABLE_FLUSH            = 12,
        NPF_MIPv6_TUNNEL_ADDR_SET               = 13,
        NPF_MIPv6_TUNNEL_TABLE_ATTR_QUERY       = 14,

        //Binding
        NPF_MIPv6_BINDING_TABLE_HANDLE_CREATE   = 15,
        NPF_MIPv6_BINDING_TABLE_HANDLE_DELETE   = 16,
        NPF_MIPv6_BINDING_ENTRY_ADD             = 17,
        NPF_MIPv6_BINDING_ENTRY_DELETE          = 18,
        NPF_MIPv6_BINDING_TABLE_FLUSH           = 19,
        NPF_MIPv6_BINDING_ENTRY_SET             = 20,
        NPF_MIPv6_BINDING_TABLE_ATTR_QUERY      = 21,
} NPF_MIPv6CallbackType_t;

/************************ Asynchrone Response ***********************/
typedef struct{
        NPF_IfHandle_t          ifHandle;
        NPF_IfID_t              ifID;
        NPF_IfErrorType_t       error;
        NPF_MIPv6CallbackType_t type;
        union{
                NPF_uint32_t                            unused;
                NPF_MIPv6_DestEntryResp_t               destResp;
                NPF_MIPV6_TunnelEntryResp_t             tunnelResp;
                NPF_MIPV6_BindingEntryResp_t            bindingResp;
                NPF_MIPV6_Dest_Tbl_Handle_Create_t
        destTbleHandleCreate;
                NPF_MIPV6_Dest_Tbl_Handle_Delete_t
        destTbleHandleDelete;
                NPF_MIPv6_Tunnel_Tbl_Handle_Create_t
        tunnelbleHandleCreate;
```

```
            NPF_MIPv6_Tunnel_Tbl_Handle_Delete_t
        tunnelTbleHandleDelete;
            NPF_MIPv6_Binding_Tbl_Handle_Create_t
        bindingTbleHandleCreate;
            NPF_MIPv6_Binding_Tbl_Handle_Delete_t
        bindingTbleHandleDelete;
            NPF_MIPV6_Dest_Tbl_Attr_Query_t          destTbleAttribute;
            NPF_MIPV6_Tunnel_Tbl_Attr_Query_t
        tunneltTbleAttribute;
            NPF_MIPV6_Binding_Tbl_Attr_Query_t
        bindingTbleAttribute;
        } u;
} NPF_MIPv6AsyncResponse_t;

/************************** Callback Data **************************/
typedef struct{
        NPF_MIPv6CallbackType_t         type;
        NPF_boolean_t                   allOk;
        NPF_uint32_t                    n_resp;
        NPF_MIPv6AsyncResponse_t        *resp;
} NPF_MIPv6CallbackData_t;

/************************ Asynchrone response **************************/
typedef struct{
        NPF_uint32_t        result;
        NPF_MIPv6DestEntry_t    entry;
}NPF_MIPV6_DestResultEntry_t;

typedef struct{
        NPF_uint32_t        numEntries;
        NPF_MIPV6_DestResultEntry_t    *entryResp;
}NPF_MIPV6_DestEntryResp_t;

typedef struct{
        NPF_uint32_t        result;
        NPF_MIPv6TunnelEntryArray_t    entry;
}NPF_MIPv6TunnelResp_t;

typedef struct{
        NPF_uint32_t        numEntries;
        NPF_MIPv6TunnelResp_t   *entryResp;
}NPF_MIPV6_TunnelEntryResp_t;

typedef struct{
        NPF_uint32_t        result;
        NPF_MIPv6BindingEntryArray_t    entry;
}NPF_MIPv6BindingResp_t;
```

```
typedef struct{
        NPF_uint32_t          numEntries;
        NPF_MIPv6BindingResp_t  *entryResp;
} NPF_MIPV6_BindingEntryResp_t;

typedef struct{
        NPF_uint32_t          result;
        NPF_MIPv6DestTableHandle_t handle;
}NPF_MIPV6_Dest_Tbl_Handle_Create_t;

typedef struct{
        NPF_uint32_t          result;
}NPF_MIPV6_Dest_Tbl_Handle_Delete_t;

typedef struct{
        NPF_uint32_t          result;
        NPF_MIPv6_TunnelTableHandle_t handle;
}NPF_MIPv6_Tunnel_Tbl_Handle_Create_t;

typedef struct{
        NPF_uint32_t          result;
}NPF_MIPv6_Tunnel_Tbl_Handle_Delete_t;

typedef struct{
        NPF_uint32_t          result;
        NPF_MIPv6_BindingTableHandle_t handle;
}NPF_MIPv6_Binding_Tbl_Handle_Create_t;

typedef struct{
        NPF_uint32_t          result;
}NPF_MIPv6_Binding_Tbl_Handle_Delete_t;

typedef struct{
        NPF_uint32_t          result;
        NPF_uint32_t          tblSize;
}NPF_MIPv6_Dest_Tbl_Attr_Query_t;

typedef struct{
        NPF_uint32_t          result;
        NPF_uint32_t          tblSize;
}NPF_MIPv6_Tunnel_Tbl_Attr_Query_t;

typedef struct{
        NPF_uint32_t          result;
        NPF_uint32_t          tblSize;
}NPF_MIPv6_Binding_Tbl_Attr_Query_t;
```

```
/************************** Event Type ****************************/
typedef enum NPF_MIPv6_Event {
        NPF_MIPV6_DEST_TBL_MISS                 = 1,
        NPF_MIPV6_TUNNEL_TBL_MISS               = 2,
        NPF_MIPV6_BINDING_TBL_MISS              = 3,
        NPF_MIPV6_TUNNEL_ENTRY_EXPIRE           = 4,
        NPF_MIPV6_DEST_ENTRY_EXPIRE             = 5,
        NPF_MIPV6_BINDING_ENTRY_EXPIRE          = 6,
        NPF_MIPV6_DEST_INCOMPLETE_ENTRY         = 7,
        NPF_MIPV6_TUNNEL_INCOMPLETE_ENTRY       = 8,
        NPF_MIPV6_BINDING_INCOMPLETE_ENTRY      = 9,
        NPF_BINDING_UPDATE                      = 10,
        NPF_BINDING_REFRESH_REQUEST             = 11,
        NPF_FAST_BINDING_UPDATE                 = 12,
        NPF_LOCAL_BINDING_UPDATE                = 13,
        NPF_HANDOVER_INITIATE                   = 14,
        NPF_HANDOVER_TO_THIRD                   = 15
} NPF_MIPv6Event_t;

/************************** Event Data ****************************/
typedef struct {
        NPF_MIPv6Event_t  type;
        NPF_IfHandle_t    handle;
        NPF_UserContext_t userContext;
        union{
                NPF_MIPv6_DestTblMiss_t         destTblMiss;
                NPF_MIPv6_BindingTblMiss_t      bindingTblMiss;
                NPF_MIPv6_TunnelTblMiss_t       tunnelTblMiss;
                NPF_MIPv6_TunnelEntryExpire_t   tunnelExpire;
                NPF_MIPv6_DestEntryExpire_t     destExpire;
                NPF_MIPv6_BindingEntryExpire_t  bindingExpire;
                NPF_MIPv6_DestIncompleteEntry_t       destIncomplete;
                NPF_MIPv6_TunnelIncompleteEntry_t     tunnelIncomplete;
                NPF_MIPv6_BindingIncompleteEntry_t    bindingIncomplete;
                NPF_MIPv6_BindingUpdate_t             bindingUpdate;
                NPF_MIPv6_LocalBindingUpdate_t        localBindingUpdate;
                NPF_MIPv6_BindingRefreshRequest_t     bindingRefreshRequest;
                NPF_MIPv6_FastBindingUpdate_t         fastBU;
                NPF_MIPv6_HandoverInitiate_t          handInitiate;
                NPF_MIPv6_HandoverToThird_t           handToThird;
        } u;
} NPF_MIPv6EventData_t;

typedef struct {
  NPF_uint32_t          numData;
  NPF_MIPv6EventData_t  dataArray[];
```

```
} NPF_MIPv6_EventArray_t;

typedef struct {
        NPF_error_t         returnCode;
        NPF_uint16_t        frameLenght;
        NPF_uchar8_t        *frame;
} NPF_MIPv6_DestTblMiss_t;

typedef struct {
        NPF_error_t         returnCode;
        NPF_uint16_t        frameLenght;
        NPF_uchar8_t        *frame;
} NPF_MIPv6_BindingTblMiss_t;

typedef struct {
        NPF_error_t         returnCode;
        NPF_uint16_t        frameLenght;
        NPF_uchar8_t        *frame;
} NPF_MIPv6_TunnelTblMiss_t;

typedef struct {
        NPF_error_t                 returnCode;
        NPF_MIPv6TunnelArray_t      tunnelArray;
} NPF_MIPv6_TunnelEntryExpire_t;

typedef struct {
        NPF_error_t                 returnCode;
        NPF_MIPv6DestEntryArray_t   destArray;
} NPF_MIPv6_DestEntryExpire_t;

typedef struct {
        NPF_error_t                 returnCode;
        NPF_MIPv6BindingEntryArray_t bindingArray;
} NPF_MIPv6_BindingEntryExpire_t;

typedef struct {
        NPF_error_t                 returnCode;
        NPF_MIPv6DestEntryArray_t   destArray;
} NPF_MIPv6_DestIncompleteEntry_t;

typedef struct {
        NPF_error_t                 returnCode;
        NPF_MIPv6TunnelArray_t      tunnelArray;
} NPF_MIPv6_TunnelIncompleteEntry_t;

typedef struct {
        NPF_error_t                 returnCode;
```

```
        NPF_ MIPv6BindingEntryArray _t  bindingArray;
} NPF_MIPv6_BindingIncompleteEntry_t;

typedef struct {
        NPF_error_t                     returnCode;
        NPF_ MIPv6BindingEntryArray _t  bindingArray;
} NPF_MIPv6_BindingUpdate_t;

typedef struct {
        NPF_error_t                     returnCode;
        NPF_ MIPv6BindingEntryArray _t  bindingArray;
} NPF_MIPv6_BindingRefreshRequest_t;

typedef struct {
        NPF_error_t                     returnCode;
        NPF_ MIPv6BindingEntryArray _t  bindingArray;
} NPF_MIPv6_FastBindingUpdate_t;

typedef struct {
        NPF_error_t                     returnCode;
        NPF_ MIPv6BindingEntryArray _t  bindingArray;
} NPF_MIPv6_LocalBindingUpdate_t;

typedef struct {
        NPF_error_t             returnCode;
        NPF_MIPv6TunnelArray_t  tunnelArray;
} NPF_MIPv6_HandoverInitiate_t;

typedef struct {
        NPF_error_t             returnCode;
        NPF_MIPv6TunnelArray_t  tunnelArray;
} NPF_MIPv6_HandoverInitiate_t;

/***************************** Functions *****************************/
typedef void (*NPF_MIPv6CallbackFunc_t) (
        NPF_userContext_t userContext,
        NPF_correlator_t correlator,
        NPF_MIPv6CallbackData_t *resp);

NPF_error_t NPF_MIPv6Register(
        NPF_userContext_t userContext,
        NPF_MIPv6CallbackFunc_t callbackFunc,
        NPF_callbackHandle_t  *callbackHandle);

NPF_error_t NPF_MIPv6Deregister(
        NPF_callbackHandle_t callbackHandle);
```

```
typedef void (*NPF_MIPv6EventCallFunc_t) (
        NPF_userContext_t userContext,
        NPF_MIPv6_EventArray_t *eventArray);

NPF_error_t NPF_MIPv6EventRegister(
        NPF_userContext_t userContext,
        NPF_MIPv6EventCallFunc_t eventCallFunc,
                        NPF_callbackHandle_t *eventHandle);

NPF_error_t NPF_MIPv6EventDeregister(
        NPF_callbackHandle_t eventHandle);

/********************* Destination Table Functions *********************/
NPF_error_t NPF_MIPv6DestTableHandleCreate(
        NPF_callbackHandle_t *cbhandle,
        NPF_correlator_t correlator,
        NPF_errorReporting_t error);

NPF_error_t NPF_MIPv6DestTableHandleDelete(
        NPF_callbackHandle_t *cbhandle,
        NPF_correlator_t correlator,
        NPF_errorReporting_t error,
        NPF_MIPv6DestinationTableHandle_t tblHandle);

NPF_error_t NPF_MIPv6DestEntryAdd(
        NPF_callbackHandle_t                *cbHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6DestinationTablehandle_t   tblHandle;
        NPF_uint32_t                        numEntries,
        NPF_MIPv6DestinationEntryArray_t    *entryArray);

NPF_error_t NPF_MIPv6DestEntryDelete(
        NPF_callbackHandle_t                *cbHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6estinationTableHandle_t    tblHandle,
        NPF_uint32_t                        numEntries,
        NPF_MIPv6estinationEntryArray_t  *entryArray);

NPF_error_t NPF_MIPv6DestTableFlush(
        NPF_callbackHandle_t            *cbHandle,
        NPF_correlator_t                correlator,
        NPF_errorReporting_t            error,
        NPF_MIPv6DestinationTableHandle_t   tblHandle);

NPF_error_t NPF_MIPv6DestAttrQuery(
```

```
        NPF_callbackHandle_t                callbackHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6DestinationTableHandle_t   tblHandle);

/********************** Binding Table Functions ************************/
NPF_error_t NPF_MIPv6BindingTableHandleCreate(
        NPF_callbackHandle_t    *cbHandle,
        NPF_correlator_t        correlator,
        NPF_errorReporting_t    error);

NPF_error_t NPF_MIPv6BindingTableHandleDelete(
        NPF_callbackHandle_t                *cbHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6_BindingTableHandle_t      tblHandle);

NPF_error_t NPF_MIPv6BindingEntryAdd(
        NPF_callbackHandle_t                *cbHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6_BindingTableHandle_t      tblHandle,
        NPF_uint32_t                        numEntries,
        NPF_MIPv6BindingEntryArray_t        *entryArray);

NPF_error_t NPF_MIPv6BindingEntryDelete(
        NPF_callbackHandle_t                *cbHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6_BindingTableHandle_t      tblHandle;
        NPF_uint32_t                        numEntries,
        NPF_MIPv6BindingEntryArray_t        *entryArray);

NPF_error_t NPF_MIPv6BindingTableFlush(
        NPF_callbackHandle_t                *cbHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6_BindingTableHandle_t      tblHandle);

NPF_error_t NPF_MIPv6_BindingAttrQuery(
        NPF_callbackHandle_t                callbackHandle,
        NPF_correlator_t                    correlator,
        NPF_errorReporting_t                error,
        NPF_MIPv6BindingTableHandle_t       tblHandle);

/********************** Tunnel Table Functions ************************/
NPF_error_t NPF_MIPv6TunnelTableHandleCreate(
```

```
        NPF_callbackHandle_t        *cbHandle,
        NPF_correlator_t            correlator,
        NPF_errorReporting_t        error);

NPF_error_t NPF_MIPv6TunnelTableHandleDelete(
        NPF_callbackHandle_t                    *cbHandle,
        NPF_correlator_t                        correlator,
        NPF_errorReporting_t                    error,
        NPF_MIPv6_TunnelTableHandle_t tblHandle);

NPF_error_t NPF_MIPv6TunnelEntryAdd(
        NPF_callbackHandle_t                    *cbHandle,
        NPF_correlator_t                        correlator,
        NPF_errorReporting_t                    error,
        NPF_MIPv6_TunnelTableHandle_t           tblHandle,
        NPF_uint32_t                            numEntries,
        NPF_MIPv6TunnelEntryArray_t             *entryArray);

NPF_error_t NPF_MIPv6TunnelEntryDel(
        NPF_callbackHandle_t                    *cbHandle,
        NPF_correlator_t                        correlator,
        NPF_errorReporting_t                    error,
        NPF_MIPv6_TunnelTableHandle_t           tblHandle,
        NPF_uint32_t                                    numEntries,
        NPF_MIPv6TunnelEntryArray_t             *entryArray);

NPF_error_t NPF_MIPv6TunnelFlush(
        NPF_callbackHandle_t                    *cbHandle,
        NPF_correlator_t                        correlator,
        NPF_errorReporting_t                    error,
        NPF_MIPv6_TunnelTableHandle_t           tblHandle);

NPF_error_t NPF_MIPv6TunnelAttrQuery(
        NPF_callbackHandle_t            callbackHandle,
        NPF_correlator_t                correlator,
        NPF_errorReporting_t            error,
        NPF_MIPv6TunnelTableHandle_t    tblHandle);

#ifdef __cplusplus
}
#endif

#endif /* NPFIPV6API_H */
```

API de gestion d'interface

```
#ifndef __NPF_IF_MIPV6_H__
#define __NPF_IF_MIPV6_H__

#ifdef __cplusplus
extern "C" {
#endif

typedef NPF_uint32_t NPF_IfID_t; /* Interface Identifier */

#ifndef __NPF_H__
/*
* Interface handle
*/
typedef NPF_uint32_t NPF_IfHandle_t;
#endif

typedef struct {
        NPF_uint32_t                    nbAddr;
        NPF_IPv6NetAddress_t            *addr;
        NPF_uint32_t                    nbMuAddr;
        NPF_IPv6NetAddress_t            *muAddr;
        NPF_uint16_t                    mtu;
        NPF_IPv6UC_FwdTableHandle_t fibHandle;
        NPF_IfIpFwdMode_t               fwdMode;
        NPF_uint8t                      functionality;
} NPF_IfMIPv6_t;

typedef enum {
        NPF_IF_TYPE_UNK=1,              /* Interface type unknown */
        NPF_IF_TYPE_LAN=2,              /* Generic LAN interface */
        NPF_IF_TYPE_ATM=3,              /* ATM interface */
        NPF_IF_TYPE_POS=4,              /* Packet over SONET interface */
        NPF_IF_TYPE_IPV4=5,             /* IPv4 logical interface */
        NPF_IF_TYPE_IPV6=6,             /* IPv6 logical interface */
        NPF_IF_TYPE_IPV6INV4=7,         /* IPv6inv4 logical interface */
        NPF_IF_TYPE_TUNNEL_IPV4=8,/* IP-in-IPV4 tunnel */
        NPF_IF_TYPE_TUNNEL_IPV6=9,/* IP-in-IPV6 tunnel */
        NPF_IF_TYPE_MIPV6=10            //MIPv6 interface
        NPF_IF_TYPE_HIGHEST=11          /* Highest defined value */
} NPF_IfType_t;

typedef struct{
        NPF_IPv6Address_t  prefix;
        NPF_int32_t        prefixLen;
```

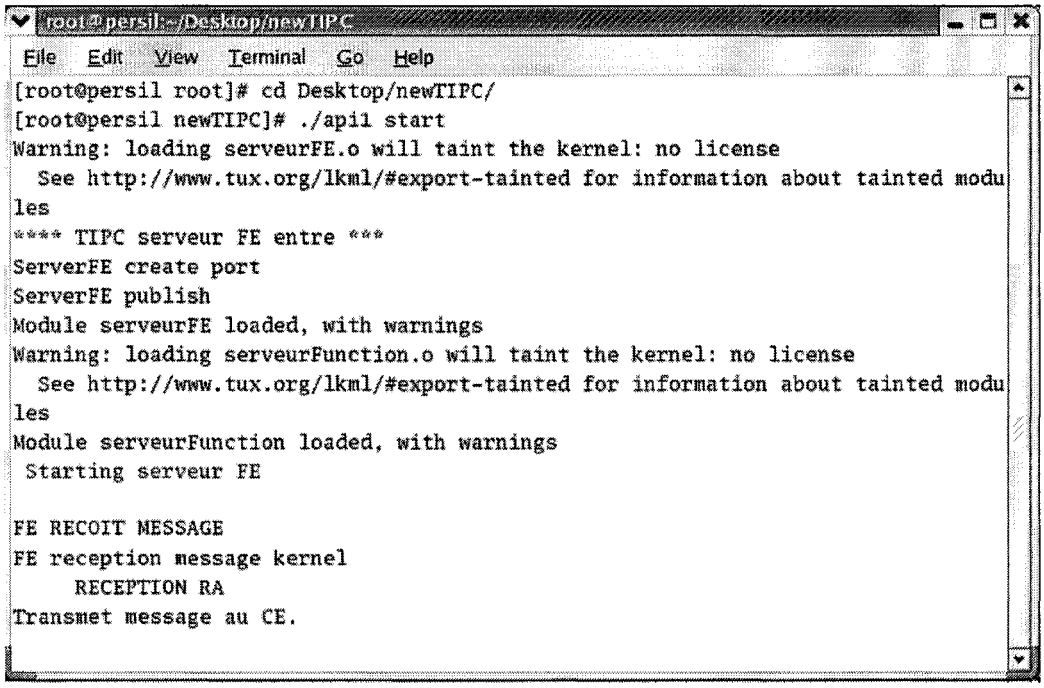
```
} NPF_MIPv6Add_t

typedef struct{
        int             result;
}NPF_MIPv6_HAEnable_t;

typedef struct{
        int             result;
}NPF_MIPv6_MAPEnable_t;

typedef struct{
        int             result;
}NPF_MIPv6_AREnable_t;

typedef enum NPF_IfCallbackType {
        ...
        ...

        NPF_IF_MIPV6_HA_ENABLE= 51,
        NPF_IF_MIPV6_HA_DISABLE= 52,
        NPF_IF_MIPV6_AR_ENABLE= 53,
        NPF_IF_MIPV6_AR_DISABLE= 54,//
        NPF_IF_MIPV6_MAP_ENABLE= 55,
        NPF_IF_MIPV6_MAP_DISABLE= 56,

        NPF_IF_MIPV6_ADDR_SET = 57,
        NPF_IF_MIPV6_ADDR_ADD = 58,
        NPF_IF_MIPV6_ADDR_DELETE = 59,
        NPF_IF_MIPV6_FIB_SET = 60,
} NPF_IfCallbackType_t;

typedef void (*NPF_IfCallbackFunc_t) (
        NPF_IN NPF_userContext_t        userContext,
        NPF_IN NPF_correlator_t         correlator,
        NPF_IN NPF_IfCallbackData_t     *resp);

NPF_error_t NPF_MIPv6Register(
        NPF_IN NPF_userContext_t        userContext,
        NPF_IN NPF_IfCallbackFunc_t     ifCallbackFunc,
        NPF_OUT NPF_callbackHandle_t    *ifCallbackHandle);

NPF_error_t NPF_IfDeregister(
        NPF_IN NPF_callbackHandle_t ifCallbackHandle);

NPF_error NPF_IfMIPv6HAEnable(
        NPF_callbackHandle_t        *cbHandle,
        NPF_uint32_t                nbInterface;
```

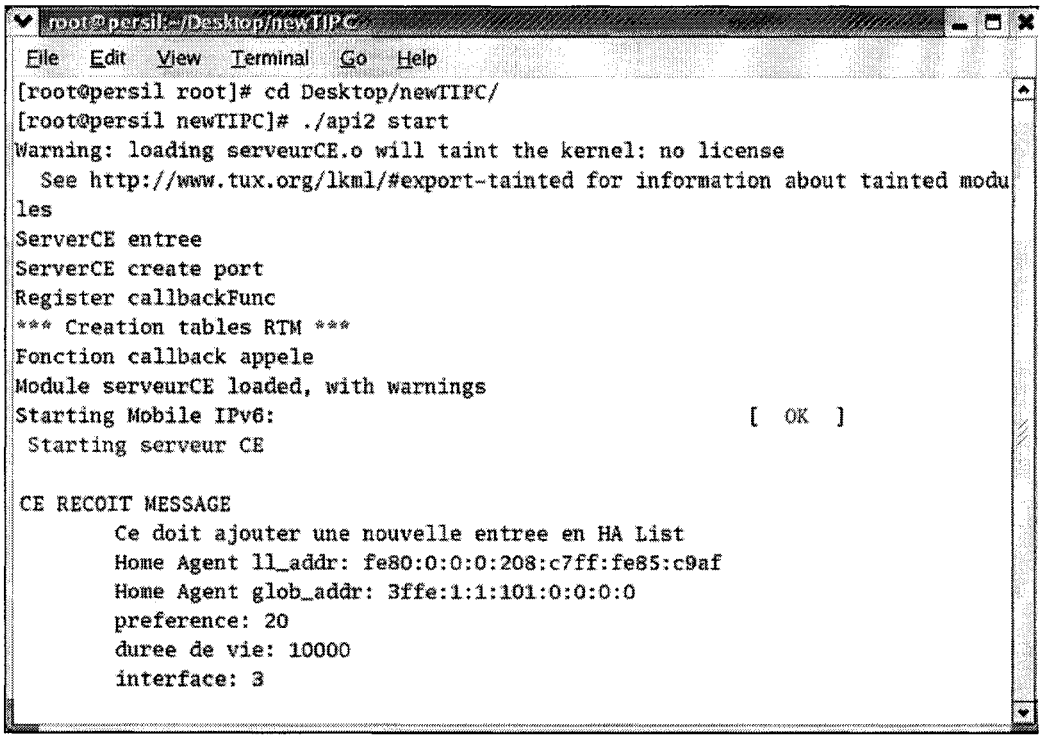
```
        NPF_ifHandle_t          *interfaceArray,
        NPF_errorReporting_t    error
        NPF_correlator_t        correlator);

NPF_error NPF_IfMIPv6HADisable(
        NPF_callbackHandle_t    *cbHandle,
        NPF_uint32_t            nbInterface;
        NPF_ifHandle_t          *interfaceArray,
        NPF_errorReporting_t    error
        NPF_correlator_t        correlator);

NPF_error NPF_IfMIPv6AREnable(
        NPF_callbackHandle_t    *cbHandle,
        NPF_uint32_t            nbInterface;
        NPF_ifHandle_t          *interfaceArray,
        NPF_errorReporting_t    error
        NPF_correlator_t        correlator);

NPF_error NPF_IfMIPv6ARDisable(
        NPF_callbackHandle_t    *cbHandle,
        NPF_uint32_t            nbInterface;
        NPF_ifHandle_t          *interfaceArray,
        NPF_errorReporting_t    error
        NPF_correlator_t        correlator);

NPF_error NPF_IfMIPv6MAPEnable(
        NPF_callbackHandle_t    *cbHandle,
        NPF_uint32_t            nbInterface;
        NPF_ifHandle_t          *interfaceArray,
        NPF_errorReporting_t    error
        NPF_correlator_t        correlator);

NPF_error NPF_IfMIPv6MAPDisable(
        NPF_callbackHandle_t    *cbHandle,
        NPF_uint32_t            nbInterface;
        NPF_ifHandle_t          *interfaceArray,
        NPF_errorReporting_t    error
        NPF_correlator_t        correlator);

NPF_error_t NPF_MIfIPv6AddrSet(
        NPF_callbackHandle_t    cbHandle,
        NPF_correlator_t if_    correlator,
        NPF_errorReporting_t    error,
        NPF_IfHandle_t          handle,
        NPF_uint32_t            nbAddr,
        NPF_ IPv6NetAddress _t  *MIPv6AddrArray);
```

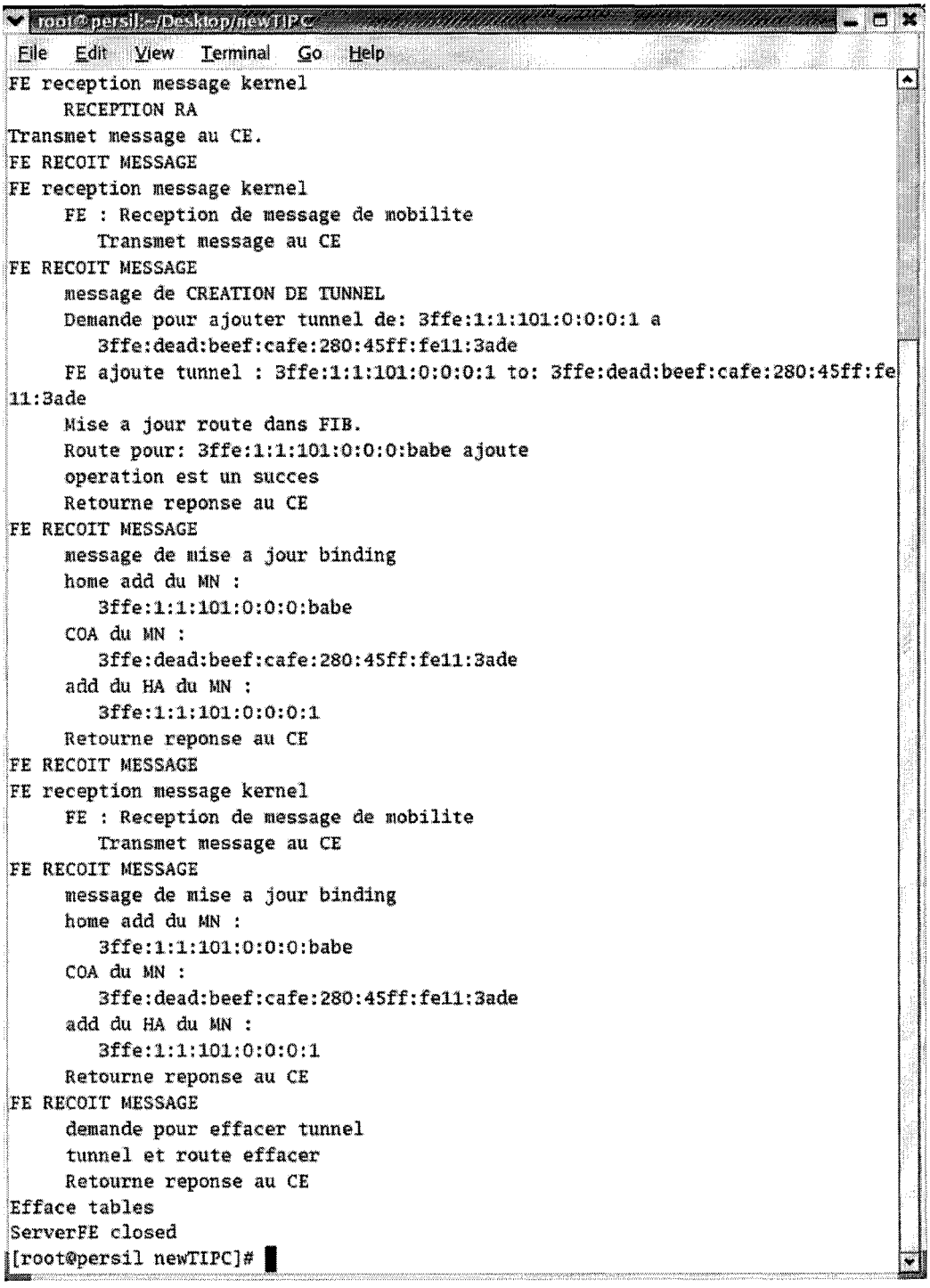
```
NPF_error_t NPF_IfMIPv6AddrAdd(
        NPF_callbackHandle_t        cbHandle,
        NPF_correlator_t            correlator,
        NPF_errorReporting_t        error,
        NPF_IfHandle_t              handle,
        NPF_uint32_t                nbAddr,
        NPF_ IPv6NetAddress _t      *MIPv6AddrArray);

NPF_error_t NPF_IfIPv6AddrDelete(
        NPF_callbackHandle_t        cbHandle,
        NPF_correlator_t            correlator,
        NPF_errorReporting_t        error,
        NPF_IfHandle_t              handle,
        NPF_uint32_t                nbAddr,
        NPF_IPv6Prefix_t            *MIPv6AddrArray);

NPF_error_t NPF_IfMIPv6FIB_Set(
        NPF_callbackHandle_t        cbHandle,
        NPF_correlator_t            correlator,
        NPF_errorReporting_t        error,
        NPF_uint32_t                n_handles,
        NPF_IfHandle_t              *handleArray,
        NPF_IPv6UC_FwdTableHandle_t   FIB_Handle);

#ifdef __cplusplus
}
#endif
#endif
```

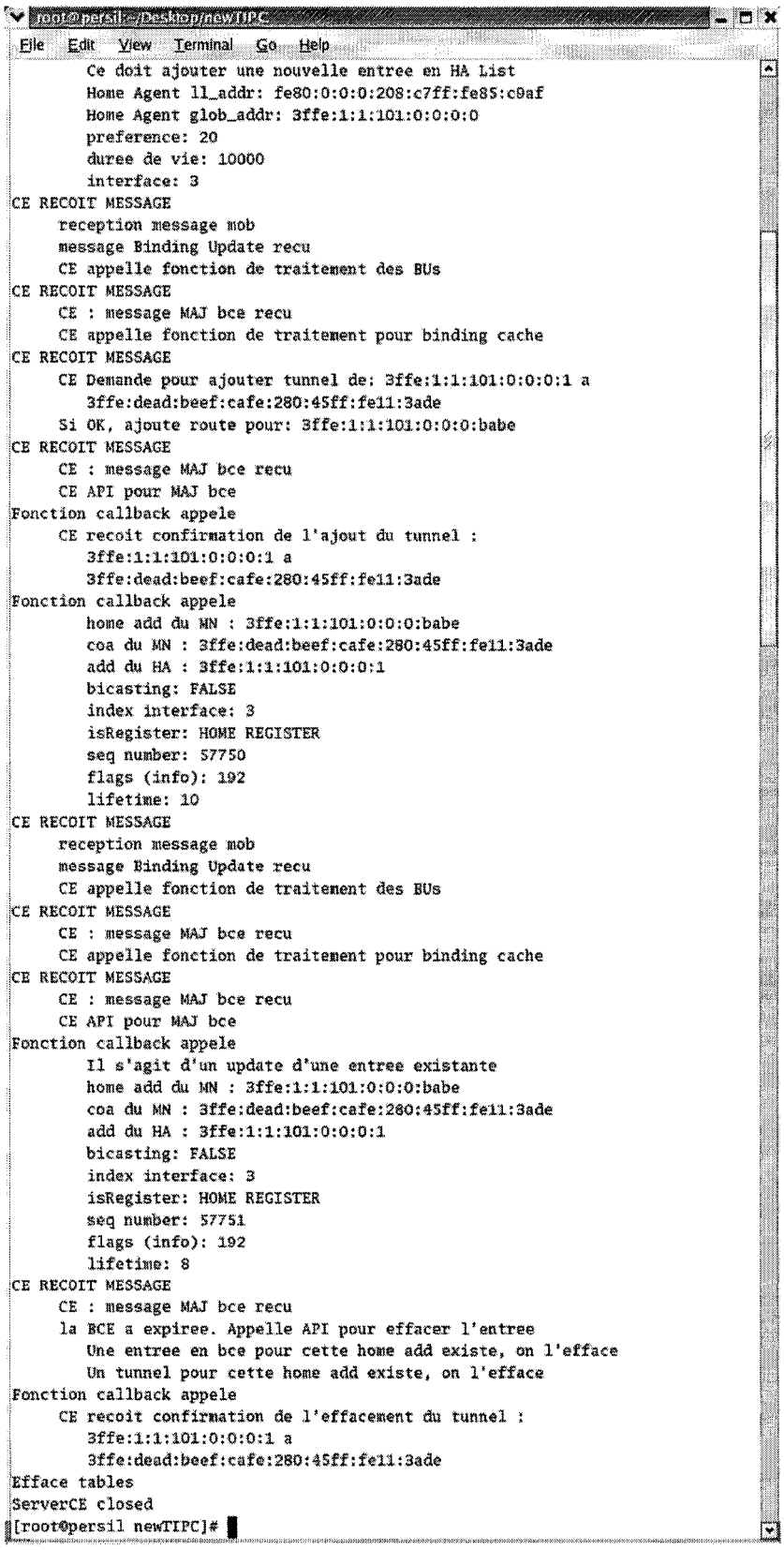
ANNEXE B

Figures




root@persil:~/Desktop/newTIPC
File Edit View Terminal Go Help
[root@persil root]# cd Desktop/newTIPC/
[root@persil newTIPC]# ./api2 start
Warning: loading serveurCE.o will taint the kernel: no license
See http://www.tux.org/lkml/#export-tainted for information about tainted modu
les
ServerCE entree
ServerCE create port
Register callbackFunc
*** Creation tables RTM ***
Fonction callback appele
Module serveurCE loaded, with warnings
Starting Mobile IPv6: [OK]
Starting serveur CE
CE RECOIT MESSAGE
Ce doit ajouter une nouvelle entree en HA List
Home Agent ll_addr: fe80:0:0:0:208:c7ff:fe85:c9af
Home Agent glob_addr: 3ffe:1:1:101:0:0:0:0
preference: 20
duree de vie: 10000
interface: 3


root@persil:~/Desktop/newTIPC
File Edit View Terminal Go Help
FE reception message kernel
RECEPTION RA
Transmet message au CE.
FE RECOIT MESSAGE
FE reception message kernel
FE : Reception de message de mobilite
Transmet message au CE
FE RECOIT MESSAGE
message de CREATION DE TUNNEL
Demande pour ajouter tunnel de: 3ffe:1:1:101:0:0:0:1 a
3ffe:dead:beef:cafe:280:45ff:fe11:3ade
FE ajoute tunnel : 3ffe:1:1:101:0:0:0:1 to: 3ffe:dead:beef:cafe:280:45ff:fe
11:3ade
Mise a jour route dans FIB.
Route pour: 3ffe:1:1:101:0:0:0:babe ajoute
operation est un succes
Retourne reponse au CE
FE RECOIT MESSAGE
message de mise a jour binding
home add du MN :
3ffe:1:1:101:0:0:0:babe
COA du MN :
3ffe:dead:beef:cafe:280:45ff:fe11:3ade
add du HA du MN :
3ffe:1:1:101:0:0:0:1
Retourne reponse au CE
FE RECOIT MESSAGE
FE reception message kernel
FE : Reception de message de mobilite
Transmet message au CE
FE RECOIT MESSAGE
message de mise a jour binding
home add du MN :
3ffe:1:1:101:0:0:0:babe
COA du MN :
3ffe:dead:beef:cafe:280:45ff:fe11:3ade
add du HA du MN :
3ffe:1:1:101:0:0:0:1
Retourne reponse au CE
FE RECOIT MESSAGE
demande pour effacer tunnel
tunnel et route effacer
Retourne reponse au CE
Efface tables
ServerFE closed
[root@persil newTIPC]#


root@persil:~/Desktop/newTIPC
File Edit View Terminal Go Help
Ce doit ajouter une nouvelle entree en HA List
Home Agent ll_addr: fe80:0:0:0:208:c7ff:fe85:c9af
Home Agent glob_addr: 3ffe:1:1:101:0:0:0:0
preference: 20
duree de vie: 10000
interface: 3
CE RECOIT MESSAGE
reception message mob
message Binding Update recu
CE appelle fonction de traitement des BUs
CE RECOIT MESSAGE
CE : message MAJ bce recu
CE appelle fonction de traitement pour binding cache
CE RECOIT MESSAGE
CE Demande pour ajouter tunnel de: 3ffe:1:1:101:0:0:0:1 a
3ffe:dead:beef:cafe:280:45ff:fe11:3ade
Si OK, ajoute route pour: 3ffe:1:1:101:0:0:0:babe
CE RECOIT MESSAGE
CE : message MAJ bce recu
CE API pour MAJ bce
Fonction callback appele
CE recoit confirmation de l'ajout du tunnel :
3ffe:1:1:101:0:0:0:1 a
3ffe:dead:beef:cafe:280:45ff:fe11:3ade
Fonction callback appele
home add du MN : 3ffe:1:1:101:0:0:0:babe
coa du MN : 3ffe:dead:beef:cafe:280:45ff:fe11:3ade
add du HA : 3ffe:1:1:101:0:0:0:1
bicasting: FALSE
index interface: 3
isRegister: HOME REGISTER
seq number: 57750
flags (info): 192
lifetime: 10
CE RECOIT MESSAGE
reception message mob
message Binding Update recu
CE appelle fonction de traitement des BUs
CE RECOIT MESSAGE
CE : message MAJ bce recu
CE appelle fonction de traitement pour binding cache
CE RECOIT MESSAGE
CE : message MAJ bce recu
CE API pour MAJ bce
Fonction callback appele
Il s'agit d'un update d'une entree existante
home add du MN : 3ffe:1:1:101:0:0:0:babe
coa du MN : 3ffe:dead:beef:cafe:280:45ff:fe11:3ade
add du HA : 3ffe:1:1:101:0:0:0:1
bicasting: FALSE
index interface: 3
isRegister: HOME REGISTER
seq number: 57751
flags (info): 192
lifetime: 8
CE RECOIT MESSAGE
CE : message MAJ bce recu
la BCE a expiree. Appelle API pour effacer l'entree
Une entree en bce pour cette home add existe, on l'efface
Un tunnel pour cette home add existe, on l'efface
Fonction callback appele
CE recoit confirmation de l'effacement du tunnel :
3ffe:1:1:101:0:0:0:1 a
3ffe:dead:beef:cafe:280:45ff:fe11:3ade
Efface tables
ServerCE closed
[root@persil newTIPC]#